Artificial intelligence and conceptual design synthesis

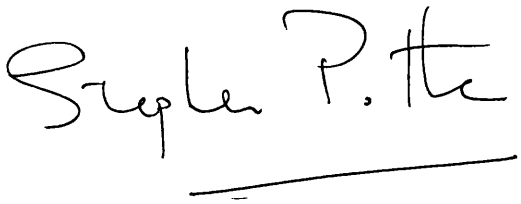
Potter, Stephen

Award date:
2000

Awarding institution:
University of Bath

Artificial Intelligence and Conceptual Design Synthesis

Submitted by Stephen Potter
for the degree of Ph.D.
of the University of Bath
2000



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Summary

This thesis is concerned with the automation of the conceptual design of engineered artefacts and systems. During the conceptual design phase, an initial solution is generated in response to the design specification; the quality of this initial solution has considerable influence on the success of the whole enterprise. In complex domains, this task is performed by an expert who has a thorough understanding of the domain and much experience of design.

As a task requiring both skill and knowledge, conceptual design has been a focus of research by the *Artificial Intelligence* (AI) community, which attempts, as a more general goal, to emulate intelligent behaviour in computer systems. While several successful automatic design systems have been developed as a result of this research, a limitation common to much of this work has become apparent. On the whole, the approaches have tended to rely on *knowledge engineering* techniques to provide the requisite knowledge. These techniques try to capture this knowledge directly from a human expert, through structured interviews or other knowledge acquisition methods. However, knowledge engineering has been found to be lacking for the capture of *heuristic* knowledge, the generalised, experience-derived 'rules of thumb' that allow experts to go about their tasks. Unfortunately, this heuristic knowledge is essential for successful conceptual design.

What is needed, then, is some approach that is able to capture these heuristics, and in such a way as to allow their subsequent re-use within computer models of the conceptual design process. The foundations of the research described in this thesis lie in the conjecture that examples of designers' work might prove a more profitable source of this heuristic design knowledge than do the designers themselves. It is hypothesised that a design example, consisting of a design specification and the corresponding design solution, contains implicitly the heuristics that the designer applied to effect the generation of solution from specification. This research concerns finding a way of exploiting this implicit design knowledge using existing AI techniques.

The working domain is that of fluid power systems. Conceptual design in this domain is a *configuration design task*, requiring the selection and connection of a set of standard domain components into a system that will meet the given specification. An *archive* of design examples in this domain has been constructed; so too has a means of describing these design problems and solutions in a computationally tractable fashion.

The application of *inductive machine learning* algorithms provides a means by which the implicit heuristics might be accessed. These algorithms have arisen from AI attempts to mimic human-like learning; each algorithm attempts to acquire, in its own particular manner, a generalised description of some concept from the evidence contained within a set of examples of that concept. As such, they would seem to hold some promise for the capture of the heuristics from examples of their application. These learned heuristics could then be incorporated within a computer system providing the appropriate mechanisms for invoking and applying this knowledge.

As will be seen, algorithms of three types - *artificial neural networks*, *classification construction* and *conceptual clustering* - have been applied to this learning task for the design problem in hand, and a number of intelligent design systems incorporating the learned heuristics have been constructed. These systems are able to generate seemingly good design solutions in response to certain design problems; however, their knowledge contains errors and is incomplete, resulting in unsatisfactory overall performance. This poor performance might be attributed to the nature of the available design examples, but is probably better ascribed to the lack of sophistication of the algorithms themselves, which display learning capabilities far below those of humans.

In response to these problems, a second approach to exploiting the heuristic knowledge embodied in the design examples has been devised and implemented. This is termed *Case-Informed Reasoning*, an adaptation of the *Case-Based Reasoning* paradigm of problem solving. This approach involves developing solutions in poorly understood domains through analogical reasoning - solutions to new problems are suggested by analogy with the manner in which similar problems were solved successfully in the past. So, rather than attempting to learn generalised heuristics from the examples, instead the information contained within specific examples is used to inform the design decisions at the time at which they are made.

This Case-Informed Reasoning method requires a greater amount of background knowledge of the domain, but it is consistent in its ability to generate good solutions. The method would seem to present a practical way of utilising the knowledge contained in the archive given the current level of understanding of design processes, and of human intelligence in general.

Acknowledgements

The author would like to acknowledge the contributions made by a number of people to the work presented in this thesis. The research was supervised by Steve Culley and Pravir Chawdhry. Steve Culley, in particular, not only encouraged the work, but also assisted greatly in the preparation of this document. Mansur Darlington was instrumental in extending the author's understanding of the subject and in the development of many of the ideas discussed here. Andy Court made a number of valuable contributions during the early stages of the research. The other members of the Engineering Design Centre at the University of Bath helped to create a friendly and stimulating atmosphere conducive to productive research, and many colleagues in the Department of Mechanical Engineering have been generous with their time and advice.

Finally, thanks are owed to the Engineering and Physical Sciences Research Council, which funded the research.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 Introduction

The term 'design' covers a wide range of activities, in fields as seemingly disparate as architectural design, graphic design, industrial process design, and mechanical engineering design. Common to all design activities, though, is the notion of *purpose*: design is a *purposeful* activity (Coyne *et al.*, 1990). In response to some expression of a desired state of affairs, the act of designing involves a conscious effort to devise the description of some artefact that, once realised, can be used to bring about this state. As implied by this, only those descriptions that are practical, inasmuch as they are realisable using available processes and technologies, will qualify as acceptable solutions. The statement of the desired state of affairs could possibly include, or be influenced by, a number of quite different factors: a description of desired functionality, aesthetic considerations, cultural and social needs and pressures, the wish to convey some particular meaning to a certain audience, and so on.

The research work reported in this thesis is concerned with mechanical engineering design. This class of design activity involves generating descriptions of mechanisms that will achieve some given goal. In general, this goal will be expressed through an abstract description of the desired artefact in terms of the physical functionality that it must provide, the performance levels it must meet and the constraints (financial, spatial, social, etc.) within which it must operate (French, 1985). Designers achieve a transformation of this design *specification* into a design solution by applying their knowledge and experience of physical principles, components, materials, and so on, as well as of the wider world. This solution will consist of a set of instructions, plans or blueprints for constructing or manufacturing the artefact.

Mechanical engineering design is not a uniform activity; the term is applied to tasks of differing complexity, requiring different skills and abilities. Pahl and Beitz (1996) identify three different general forms of mechanical engineering design: *original design*, *adaptive design* and *variant design*. Original design involves developing new solution principles (i.e., notions of how to realise the desired functionality, etc.) to solve the design problem. For adaptive design, the embodiment of established solution principles is modified to meet the

design specification. Variant design involves altering the dimensions or parameters of existing designs to satisfy the specification.

Some of these design tasks, especially the original and adaptive forms, can be very demanding of the designer. The expression of the design specification can be wide-ranging, in both content and style, and the designer must be able to understand it and grasp its implications for a valid design solution. Typically, valid solutions will constitute only a small fraction of the great number of potential solutions available to the designer – if an acceptable one is to be found efficiently, effective ways of eliminating unsatisfactory candidates from consideration and evaluating those that remain must be applied (Pugh, 1991). Generally speaking, there will be no simple algorithm that can be applied to produce correct designs every time; instead, designers must rely on their abilities, knowledge and experience to make the shrewd decisions which will resolve the design process satisfactorily.

1.1 Computer Methods in Engineering Design

A good design can do much to ensure the eventual success of an artefact in the market place; on the other hand, a poor design can all but condemn the artefact to failure. Moreover, the quality of the design work can also have a significant impact on the success of the organisation as a whole, since this success is a factor of that of the organisation's products. When this relationship is acknowledged, designers are highly valued employees.

Their value is further increased when it is recognised that not everyone possesses the characteristics that make a good designer – designers tend to be creative, intelligent people. Furthermore, it can take a considerable amount of time to acquire the necessary understanding and experience of the domain (up to ten years, suggests Ullman (1997)).

This combination of factors ensures that designer expertise is a valuable, and often scarce, commodity. As a consequence, with the advent of powerful computers has come the idea of satisfying this demand for design expertise through the construction of computer-based design tools. Traditionally, designers have used their knowledge, or that found in books, along with tools such as the slide rule and the drawing board. In contrast, modern computers have the potential to be "a single medium both for representing knowledge and for carrying out the operations in design" (Coyne *et al.*, 1990), offering an environment in which the design task is facilitated or, perhaps, even automated, either partially or in its entirety.

Indeed, computers have been used to great advantage in engineering design for a number of years. Computer-aided drafting packages are widely used, as are finite element analysis and

simulation tools. These systems are primarily design assistants, performing tedious analytical computation or providing working environments conducive to the creativity of human designers.

In this research, however, the emphasis lies on automating the design process itself, that is, on investigating the construction of a computer tool containing design knowledge which can be applied to create the design solutions themselves. Among the putative benefits of such a knowledge-based design tool are the following:

- Once built, a computerised designer would not be prone to human error or forgetfulness, and so, would be able to produce designs of consistently high quality.
- Design expertise would no longer be lost to the organisation when human designers retire or move to other companies.
- Computer files and memory can be duplicated with ease: this would allow the design expertise to be distributed throughout the organisation, and internet facilities could enable it to be transferred rapidly to the remote location at which it is needed.

Encoding the design process as a conventional computer program would require a well-defined, explicit algorithm for the task. However, for all but the most trivial of design tasks, no such algorithm will be available. An alternative approach is necessary. Since *Artificial Intelligence* (AI) techniques are expressly concerned with capturing ill-defined, flexible, *human-like* performance, it is to these that most research into design automation has looked. Indeed, from the 1970s, these techniques have been applied, with varying degrees of success, in a number of attempts to automate design. These implementations, which will be discussed in a later chapter, have served to highlight the overriding importance of having good design knowledge – and also the difficulties involved in acquiring this knowledge and expressing it on computer.

This research is concerned with addressing this 'knowledge problem'. The basis of the work lies in the identification of a possible source of design knowledge other than designers themselves – namely, examples of their work. Designers use their knowledge and experience to produce good designs: as a consequence, this knowledge and experience can be considered 'implicit' in the final design. This insight has encouraged an investigation into applying AI techniques to capture the knowledge from this source, and then to embody it within appropriate inference and control mechanisms to form an automated design system.

This work is focused on the *conceptual design* task, an early phase of the design process during which a design specification is translated into a basic solution (or solutions), in which the basic principles underpinning its operation have been established. Later stages of the

design process will embellish this solution to the extent that it can be realised as a concrete artefact. The domain chosen for this research is that of fluid power. Conceptual design problems in this domain are *configuration design* tasks. To solve such problems, designers must generate a description of some system that is constructed from existing elements – in this case, hydraulic components and pipes. Typically, the design specification will describe the force or motion that this system must be capable of imparting to a particular object or location. Conceptual design, fluid power and configuration design will all be described in detail later in this thesis.

The current level of understanding of complex design tasks, such as the conceptual design of fluid power systems, and of the approaches to solving them is quite low. As a consequence, the wholly automatic designer is unlikely to become a reality for some time, if ever, and the research presented herein should be viewed in this light. Nevertheless, a beneficial side-effect of this, and indeed, any, investigation into design using computers is that it forces a degree of reflection upon the design process, which may provide useful insights into how its performance can be improved, whether this performance be by human or computer.

In the remainder of this chapter, the mechanical engineering design process is discussed in detail, as is the particular class of problem of which fluid power systems design is a typical example – configuration design. The field of AI is introduced, and the ideas of automating intelligent behaviour that it embraces allow an explicit statement of the objectives of the research to be made.

1.2 The Mechanical Engineering Design Process

Over the last few decades, a number of models of the mechanical design process have been proposed. In the main, these are *prescriptive* models; they prescribe a general framework, which, according to their authors, if adopted would lead to better overall design performance. The models are not at the level of precise, algorithmic descriptions of how to solve design problems. Rather, they are at a higher level, describing the nature of the tasks which need to be done, if the design is to proceed in the manner most conducive to generating optimal, or near optimal, solutions, and in less time. According to Pahl and Beitz (1996), the purpose of such a model is to encourage the process to be "more logical, more sequential, more transparent, and more open to question". A good model should remove some of the 'mystery' surrounding design, so as to promote good design practice in general, and assist in reusing design experiences and analysing design failures. The objective is not, however, to remove the element of intuition from the task; rather, it is to provide a structure within which intuition is focused and applied productively.

Research devoted to the development of such models has been particularly strong in Germany. This work has resulted in, amongst others, a model endorsed by the Verein Deutscher Ingenieure (VDI), the German professional engineers' association, which promotes a systematic, industry-independent approach progressing through seven general stages, from the initial clarification of the design task through to the proper documentation of the devised solution (VDI 2221, 1987). (A British standard (BS 7000, 1989) with similar aims has also been published.) A survey of the major models may be found in (Finger and Dixon, 1989).

Another of the models developed in Germany, the systematic design methodology of Pahl and Beitz (1996), will now be discussed as illustrative of these models. It is widely referenced in design literature and has elements analogous to those found in many of the other models. It will be used throughout this work to describe the various stages of the design task and to set this research in a wider context. The methodology is prescriptive, but based upon cognitive studies of effective problem-solving behaviour.

There are four main stages to the design methodology (Figure 1). Each stage involves a cycle of defining the essential problem, creating potential solutions, evaluating these and choosing from amongst them. The four stages are as follows:

1. **Planning and clarifying the task.** Assuming that the need for some artefact has been recognised, this stage involves collecting information about the requirements that must be fulfilled by the artefact (what it must do), and the constraints within which it must operate. This information is then used to drive the following stage. The initial statement of the problem, the *design brief*, as provided by the client, can be vague, offering little information. It is the job of the designer to develop the brief into a more complete description of the problem incorporating a statement of the design problem proper, the limitations placed upon the solution and the criteria for recognising a successful solution. Hence, the problem should now be stated in terms that the designer understands and can relate to his/her field of expertise. This statement of the *design specification*, as it will be referred to in this thesis, can then be used to drive the subsequent design stages. However, it should not be considered to be definitive: later work may reveal inadequacies that can only be resolved through additional consultation with the client.
2. **Conceptual design.** This stage is one of determining the principles upon which a design solution is to be founded, and then, by making preliminary decisions about components, materials, layout, etc., 'concretising' these into a working solution (or solutions). At this point, evaluation on the basis of economic and technical feasibility can be carried out.

The research in this thesis is concerned with the performance of this stage of the design process; consequently, it is considered in greater detail in the following section.

3. **Embodiment design.** Here, the best conceptual design solution is developed further. The choices that have been made are elaborated, adding more precise information about layout and sizing. The additional information produced allows a further evaluation of the design to be made.
4. **Detail design.** The precise dimensions, materials, component models, and so on are chosen. This allows exact costing of the solution, allowing its feasibility to be accessed. If acceptable, detailed production drawings and documentation can now be produced.

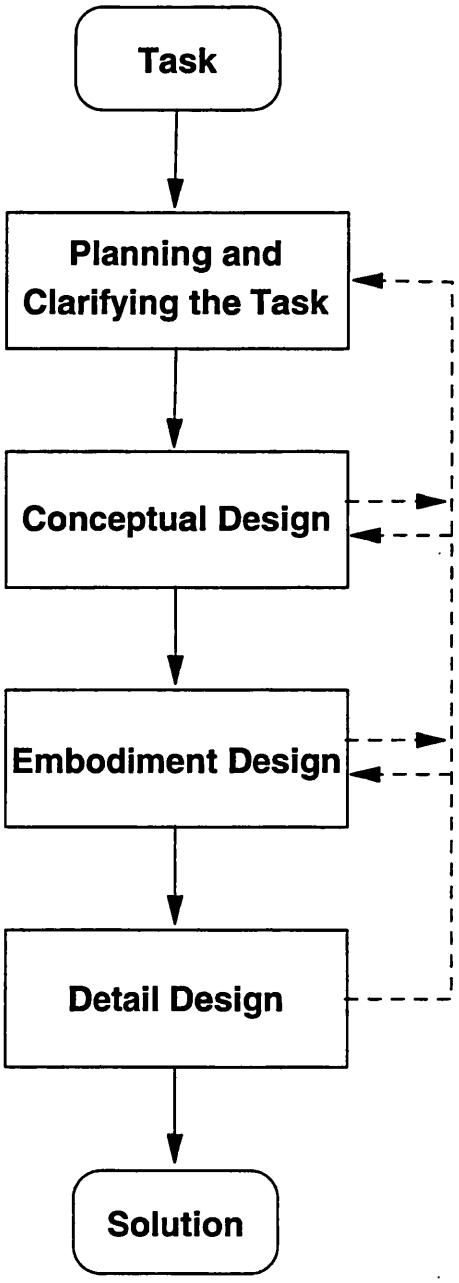


Figure 1. Model of design process, after Pahl and Beitz (1996).

It is not always possible to draw a clear distinction between the end of one phase and the beginning of the next, and in some design tasks certain steps may be of lesser or greater complexity. It is important to note that the methodology at this high level is an iterative one. The failure to successfully complete any of the later stages will entail the return to a previous stage, to generate additional information, or revise the choices that were made. A failure at the planning and clarification stage may indicate that the initial idea or specification is not feasible or practicable.

This research work is concerned with the conceptual design phase, the phase that:

> *"makes the greatest demands on the designer, and where there is the most scope for striking improvements...and where the most important decisions are taken"* (French, 1985, p. 3).

1.3 Conceptual Design

The task of the designer during the conceptual design phase is to translate the design specification into one or more initial solution concepts. According to Ullman (1997), a concept is "an idea that is sufficiently developed to evaluate the physical principles that govern its behavior." He goes on to say:

> *"Confirming that the proposed product will operate as anticipated and that, with reasonable further development, it will meet the targets set is a primary goal in concept development. Concepts must also be refined enough to evaluate the technologies needed to realise them, to evaluate the basic architecture (i.e., form) of them, and, to some limited degree, to evaluate their manufacturability. Concepts can be represented in a rough sketch or flow diagram, a proof-of-concept prototype, a set of calculations, or textual notes – an abstraction of what might some day be a product. However a concept is represented, the key point is that enough detail must be developed to model performance so that the functionality of the idea can be ensured."* (p. 120)

French (1985) refers to the results of the conceptual design stage as *schemes*. He defines a scheme to be:

> *"an outline solution to a design problem, carried to a point where the means of performing each major function has been fixed, as have the spatial and structural relationships of the principal components. A scheme should be sufficiently worked out in detail for it to be possible to supply approximate costs, weights, and overall dimensions, and the feasibility should have been*

> *assured as far as circumstances allow. A scheme should be relatively explicit about special features or components but need not go into much detail over established practice."* (p. 1)

It should be evident that the outcome of the conceptual design phase will differ according to the design problem being addressed, and that the phase has no definite start and end points. However, in general terms, in accordance with the descriptions given above, the conceptual design phase is completed when an initial design that incorporates 'working principles' or physical solutions to all the 'essential' features of the problem has been proposed, and evaluated to be acceptable and feasible (Pahl and Beitz, 1996).

Most models of conceptual design found in the literature (e.g., (Pahl and Beitz, 1996), (Pugh, 1991)) emphasise the iterative, cyclical nature of this phase of design. Initial schemes are synthesised and then analysed to judge their feasibility and select the best; those selected are then developed further, and evaluated, and so on, until a satisfactory final scheme is produced. This type of process has been observed in the work of successful designers, and so is prescribed as the best way for other human designers to approach the task, encouraging, as it does, incremental development of solutions, and frequent appraisal of the generated ideas to prevent much wasted effort. These methods are ones that, hopefully, will assist the majority of designers to arrive at solutions to the majority of problems.

However, for the purposes of this work, it is considered that these should be understood as *human* models of the process – they take into account human capabilities, and human frailties: working and short-term memory limitations, the tendency to make errors, and so on. They are not necessarily models of the process as might be performed by computer. Using comprehensive (and, of course, correct) design knowledge, and with a complete specification of the design problem, the task could conceivably be performed in one 'step', with the synthesis of an appropriate scheme.

This single step model of the process is that adopted in the work described here. As a result, this research is wholly concerned with the automation of design scheme *synthesis*. For synthesis, design textbooks (e.g., (Pahl and Beitz, 1996), (Ullman, 1997), (Cross, 1994)) suggest a range of techniques, from examining the existing literature in the domain and searching for analogous solutions in other domains, to organising group brainstorming sessions. The TRIZ methodology (Altshuller, 1984), the "Theory of Inventive Problem Solving", draws upon the principles expressed in patents across a range of domains to suggest new or better ways of solving design problems. However, while these approaches may provide the necessary stimulus to the designer, or situations conducive to idea

development, they *support* synthesis, rather than *accomplish* it. None of these approaches can guarantee a valid solution as an outcome: ultimately, synthesis is dependent on the knowledge and experience of the designer.

Consequently, the emulation of conceptual design synthesis on computer involves the expression and capture of appropriate knowledge, and the subsequent use of this knowledge within computer models of the design process. The intention is to produce a system that, in response to some adequately stated specification, is able to automatically synthesise schemes of a consistently high quality.

The particular conceptual design task which this research concerns is that of fluid power systems. This is a *configuration design* task; schemes are descriptions of connected systems of existing domain fluid power components. A later chapter is devoted to fluid power systems and their design; this chapter, however, continues with a discussion of configuration design in general.

1.4 Configuration Design

Mittal and Frayman (1989) define configuration design as follows:

> *"Given: (A) a fixed, pre-defined set of components, where a component is described by a set of properties, ports for connecting it to other components, constraints at each port that describe the components that can be connected at that port, and other structural constraints; (B) some description of the desired configuration; and (C) possibly some criteria for making optimal selections.*
>
> *"Build: One or more configurations that satisfy all the requirements, where a configuration is a set of components and a description of the connections between the components in the set, or, detect inconsistencies in the requirements."*

They go on to identify three particular characteristics of this family of design tasks:

- The components that are available are pre-defined: new components cannot be designed in the course of the task;
- The manner in which components can be connected is also pre-defined and fixed;
- In addition to specifying the selected components, a solution must contain the information about how they are to be connected.

Although this definition is widely referenced, it is not as general as it might be. For instance, as Brown (1998) remarks, their use of 'connect' and 'ports' seems to have been influenced by the domain in which Mittal and Frayman were working, namely the configuration of computer systems. The components need not interact via physical connections (but instead through magnetic fields, for example), and, indeed, the components themselves need not be physical – planning tasks, involving the allocation of resources can usefully be viewed as configuration design.

In an attempt to disambiguate and describe configuration tasks, Wielinga and Schreiber (1997) introduce three dimensions along which they characterise a configuration task: the task is described along each dimension by the appropriate choice of one of three possible categories.

The first dimension is that of *components*. The domain elements available to perform the overall task are described according to one of the following categories:

- The task involves a fixed set of components (making the task, in effect, one of determining their optimal (or, failing this, good) arrangement). This is labelled category *c1*.
- A fixed set of parameterised components (*c2*). In addition to their arrangement, suitable parameters must be allocated to the components to complete the design.
- A set of types of parameterised components (*c3*) – any number of a particular type of component may be employed to solve the problem.

The second dimension, that of the available *assembly*, can be described as:

- A fixed assembly is given (*a1*) – that is, the components and connections are already defined;
- A skeleton assembly is provided (*a2*): certain components and relationships are pre-defined; other components must be selected and positioned within the framework to complete the task;
- The choice of assembly is free (*a3*): there is no prior constraint placed upon the choice of components, nor on their arrangement.

The final dimension is that of *requirements and constraints*:

- The requirements and constraints refer to, and are directly satisfied by, individual components (*r1*);

- The requirements and constraints are *incrementally applicable* (*r2*): that is, they refer to sub-assemblies and groups of components.
- The requirements and constraints are functional or *globally applicable* (*r3*). In other words, they refer to the functionality of the complete system, and, as such, the success or otherwise of the design process can only be determined when the entire design has been specified.

These are summarised in Table 1. Various combinations of these categories describe quite different tasks, ranging from the simplest, the local verification of a pre-arranged, fixed set of components (a <c1, a1, r1> problem), to the most complex, the free assembly of an unlimited number of components in response to globally defined requirements (<c3, a3, r3>). So, the expression 'configuration design' actually covers a number of, at times quite different, tasks, and not all configuration design tasks are necessarily conceptual design tasks. As will be seen, the conceptual design of fluid power systems is a complex <c3, a3, r3> task, since it rests upon devising solutions that will provide the basic required functionality, as stipulated in the design specification. On the other hand, a <c2, a1> task, which would involve parameterising a fixed set of components in a fixed assembly, would be an example of a *variant* or *detail* design task, and a <c1, a3> task is a layout or scheduling problem.

Components	***Assembly***	***Requirements and Constraints***
C1. Fixed set	A1. Fixed	R1. Locally applicable
C2. Fixed set, parameterised	A2. Skeleton provided	R2. Incrementally applicable
C3. Set of types of parameterised components	A3. Free	R3. Globally applicable

Table 1. A summary of the dimensions for describing configuration design problems.

As noted above, conceptual design tasks are dependent on the knowledge and experience of the designer, with, in general, no simple algorithm for scheme synthesis available, and the configuration design of fluid power systems is no exception. As a consequence, in order to automate fluid power systems design, it would seem to be necessary to look to AI techniques.

1.5 Artificial Intelligence

Artificial Intelligence (AI) is the study of making machines do the sort of things that are achieved by human minds. Areas of potential interest for AI might include, for example,

understanding a foreign language, recognising faulty components on a conveyor belt, learning how to predict engine failure, playing chess, or planning a route between two cities. Typically, the 'machines' in question are digital computers, but it is important to realise that AI is not the study of computers, nor of computer programming. Rather, it is the study of intelligence, both in thought and in action, and the computer is used as the tool for testing theories of intelligent behaviour. This requires computers to be viewed as something more than 'number-crunching' machines; usually, they are considered to be symbol manipulators and, as a result, most AI theories of intelligent behaviour are described in terms of symbol manipulation (Newell and Simon, 1976). However, the use of a computer in this fashion entails making certain assumptions about the nature of intelligence. Chief amongst these is the assumption that intelligence *can* be expressed in this manner and on these types of machines – which, as it is an assumption that not everyone is willing to make, can undermine the claims of AI to be the study of intelligent behaviour *in general.*

Charniak and McDermott (1985) trace the origins of AI to the Dartmouth Conference of 1956; however, many of the tools and theories of AI research have had a much longer existence, being appropriated from the fields of logic, psychology and philosophy. Initial expectations were optimistic, to say the least, with most researchers underestimating the complexity of intelligent behaviour and the difficulties of emulating it. Indeed, the major *success* of AI to date might be the extent to which it has highlighted the vast amount of resources and background knowledge that are necessary for even what appears to be the simplest intelligent behaviour. With the computer as a tool to test its theories, AI practice insists that models be fully and precisely defined, where otherwise they might remain vague; this precision reveals the extent and viability of the assumptions made, and, ultimately, the value of the model.

Current AI research comprises work in a number of distinct areas, including (Boden, 1987):

- machine vision (the ability to process visual information, and recognise objects);
- natural language understanding (the parsing and comprehension of human speech and writing);
- problem solving (which includes tasks such as planning itineraries, designing artefacts and games playing);
- machine learning (the ability to gain knowledge through interpreting data or facts);
- robotics (which typically involves integrating problem solving with intelligent action).

Perhaps the fundamental concept to have arisen from AI research, and which underpins the different areas of research, is the overriding importance of knowledge and of its representation (Boden, 1987). A representation might be thought of as a stylised version of

the world (Charniak and McDermott, 1985), and dictates the form, construction, and manipulation of knowledge. In other words, it is not merely the content of the knowledge, but also the way in which it is stored and arranged, which makes intelligent behaviour possible. As Winston (1993) remarks, "good representations are the key to good problem solving". Accordingly, a number of different models have been proposed for knowledge representation, some better suited to certain tasks than to others. Knowledge and its representation will be discussed from an AI perspective in chapter 3.

Hence, so as to automate any task requiring intelligence – such as design synthesis - it is necessary to incorporate within some system the knowledge, represented appropriately, which will allow the task to be performed. However, for many tasks, the acquisition of this knowledge has been found to present a major problem.

1.5.1 The Knowledge Bottleneck Problem

The knowledge and its representation must be acquired from some, presumably human, source, and expressed in a suitable fashion. The conventional AI approach to acquiring the knowledge for any intelligent computer system is through a process of *knowledge engineering*, which usually consists of a series of interviews with experts in the attempt to formulate the knowledge. As will be discussed in detail in chapter 3, this process is protracted and often produces less than satisfactory results, leading to what has been termed the *knowledge bottleneck* (Lenat, 1983) in intelligent system development.

The research described in this thesis is founded on the recognition of an alternative source of the knowledge required for an automatic conceptual design system: it is postulated that design synthesis knowledge resides in existing examples of designers' work. The ability to mine this source could lessen, or remove entirely, the need for consultations with humans, and thereby alleviate the difficulties of the knowledge bottleneck.

1.6 Research Aims

The principal research objective of this work can be summarised in the form of the following hypothesis:

> *Artificial Intelligence techniques can be used to access the design synthesis knowledge implicit in previous designs and then re-apply it to produce conceptual solutions to new design problems.*

Implied by the hypothesis are the following objectives:

- The formulation of methods for representing design specifications and design solutions;
- The collection of an archive of previous (successful) designs within the chosen domain;
- The selection or development of AI methods for acquiring the knowledge implicit in the archive;
- The postulation of computer models of the process, within which this knowledge can be applied to new specifications so as to devise new schemes.

In particular, the research involves constructing some system that will automatically produce conceptual designs of fluid power systems. Accordingly, synthesis knowledge for this task is considered to reside in examples of fluid power design problems and their corresponding solution systems. Some manner of describing these problems and solutions, which captures the essentials of this conceptual design task and which is tractable in computer terms, must be devised. Techniques for mining the knowledge in the examples must then be suggested, along with models of the fluid power system design process in which this knowledge is used. It is hoped that, in general, the findings will be applicable to similar configuration design tasks in other domains.

1.7 The Structure of this Thesis

This first chapter has introduced the design task, and has, briefly, put forward the case for using computer-based methods, either to assist the designer in this task, or else to automate phases of the task. One model of the design process, that of Pahl and Beitz, a widely referenced model, has been described in greater detail. This thesis concerns the *conceptual design* phase of this model, during which initial solutions are postulated, and, in particular, ways in which conceptual design can be automated. Specifically, it is the conceptual design of fluid power systems, a *configuration design* task, which is the focus of the research.

The chapter also contains a brief discussion of the field of Artificial Intelligence, which offers techniques for implementing intelligent behaviour, such as that displayed by human designers. Previous approaches to design automation using AI methods have encountered the *knowledge bottleneck* problem – the desire to overcome this difficulty provides the motivation for this research.

Figure 2 shows the overall chapter structure of this thesis. Chapters 2, 3 and 4 consist, in the most part, of background and review material.

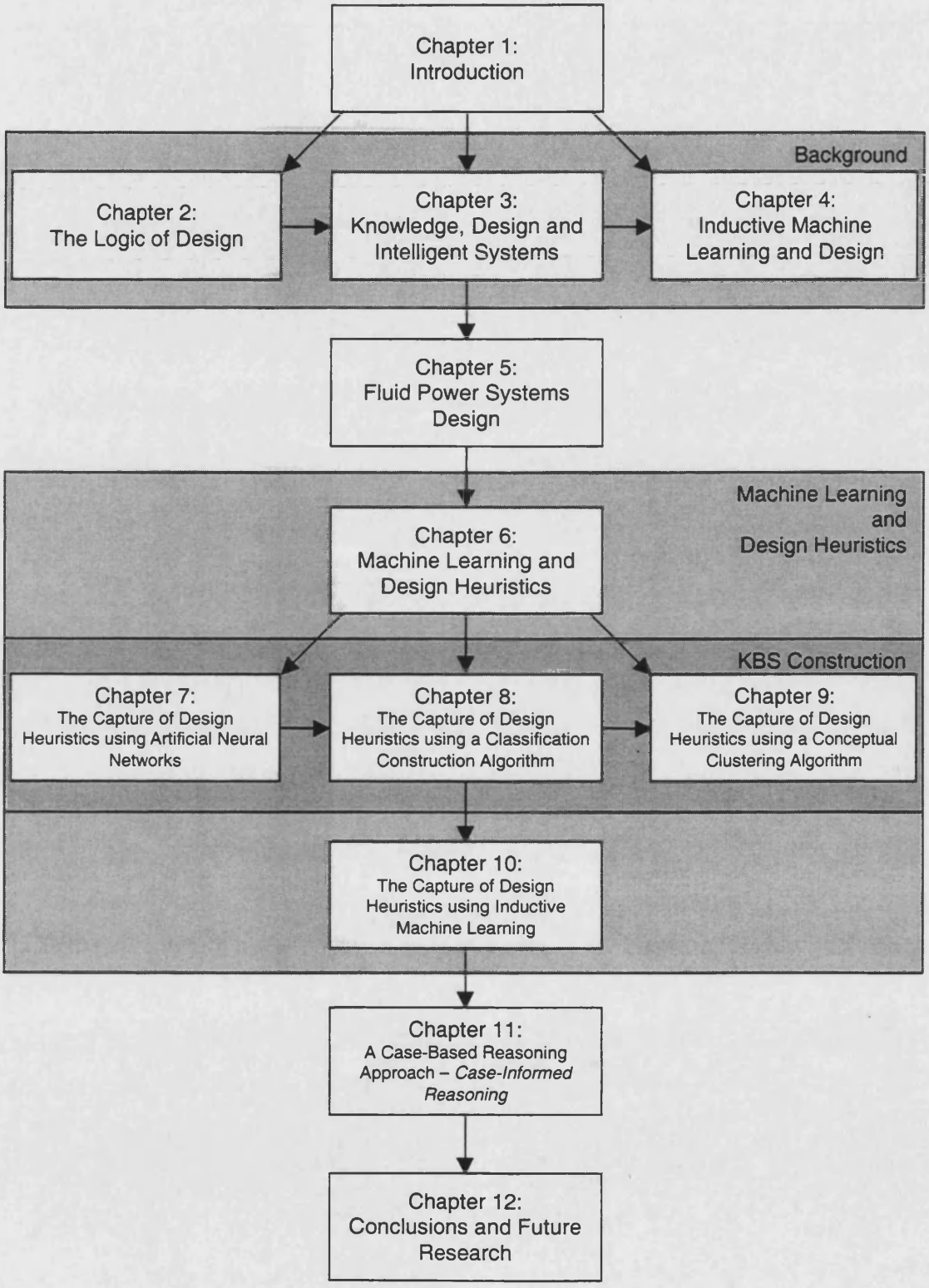


Figure 2. Structure of the thesis.

Chapter 2 discusses the logic of design: the identification and emulation of the sort of reasoning which permits successful design solutions to be produced is central to this work. This chapter includes an original extension of the existing logic to explain how original designs may be produced for configuration problems.

Chapter 3 examines in greater detail the nature and use of knowledge in AI and in design. An essential component of design expertise is *heuristic* knowledge, rules of thumb based on experience, which is used to bring about the sort of reasoning which, according to the previous chapter, characterises design synthesis. To automate this reasoning, then, it is necessary to represent these heuristics in computer systems. However, the standard approach to the acquisition of these heuristics, *knowledge engineering*, has been found wanting, for the reasons discussed.

This chapter also reviews some of the previous attempts at design automation. In general, these attempts fall into one of two categories: *knowledge-based systems* and *case-based reasoning systems*. The former rely on explicit heuristics, while in the latter, some of the heuristic knowledge is stored in the implicit form of examples of previous design episodes.

Chapter 4 introduces the sub-field of AI that is *inductive machine learning*, which has been suggested as a potential solution to the knowledge bottleneck problem in the acquisition of heuristics. Several previous applications of machine learning to design processes are discussed.

Chapter 5 and the subsequent chapters constitute the bulk of the original work reported in this thesis.

Chapter 5 deals in greater detail with the conceptual design of fluid power systems. This is used as the basis for the development of representations of this design task in computational terms. As will be seen in later chapters, these representations have been used in a number of design systems that have been developed. The chapter also discusses an *archive* of design examples that has been constructed as a source of heuristics for this research.

Chapter 6 introduces the idea of capturing design heuristics from the archive of examples using inductive machine learning. These heuristics would then be incorporated within a knowledge-based system (KBS) to give a complete automated tool for this particular design task. A method of testing these systems so as to get some measure of their comparative value is also presented.

Chapters 7, 8 and 9 discuss in detail a number of systems that have implemented in this fashion. In effect, these represent the experimental test-beds for the work.

Chapter 7 describes three systems developed using *artificial neural networks*, a particular form of inductive machine learning.

Chapter 8 presents a system that has been constructed using a *classification construction* machine learning algorithm.

Chapter 9 describes a system built using a *conceptual clustering* machine learning algorithm.

Chapter 10 summarises and discusses the systems that have been developed, and evaluates the usefulness of inductive machine learning for the capture of design heuristics. The limitations of this approach have led to the investigation of an alternative approach to exploiting the archive of design examples

Chapter 11 describes the *Case-Informed Reasoning* method, a variation on the basic Case-Based Reasoning approach. The system implementing this method has been found to be able to automatically produce consistently good designs; Case-Informed Reasoning seems to offer a practical solution to the problems surrounding the capture and application of heuristic knowledge for design problems such as the one considered here.

Chapter 12 summarises the findings and identifies several areas of potential future research.

2 The Logic of Design

Notwithstanding its ill-defined, experiential nature, design *is* a rational process: design does not proceed by chance and "designs...do not come out of the blue" (March, 1976). Designers are highly trained, highly practised individuals, who, through their skills, are able to devise a design in response to some specification. When coming to automate this process, it would seem to be essential to have some model of the logical processes that underpin this translation from specification to design solution.

It is customary to recognise three principal modes of inference: *deduction*, *induction* and *abduction*. This chapter begins with a discussion of each of these. This is followed by the description of a logical framework for solving design problems, in which the reasoning proceeds through the successive application of each type of inference. As will be shown, this framework has some serious shortcomings as a general description of design synthesis, but, nonetheless, with the addition of some further assumptions, it provides a useful model of the nature of the reasoning that needs to be emulated to produce an automated system for configuration design.

Throughout this discussion, it is important to bear in mind that logic does not describe how to reason to solve some problem; rather, it provides a model to which to this reasoning should conform. As Zeng and Cheng (1991) note, the formalisation of any problem is necessary before it can be mechanised, and, furthermore, that:

> *"...the formalization of a problem solving process is composed of two parts with the first one being the logic of the process and the second the knowledge based on the logic...."*

It is the knowledge of the process that allows the problem to be solved.

2.1 Deduction

A deductive reasoning process is one in which a conclusion is drawn from a set of premises in such a manner that the inference is logically valid. In other words, if the premises are true, the conclusion must also be true.

Consider the following example of a categorical syllogism, a typical form of deductive reasoning (taken from Roozenburg (1993)):

IF an artefact is made of aluminium	
THEN the artefact will not rust	*rule*
artefact x is made of aluminium	*proposition*
———	*(therefore)*
artefact x will not rust	*conclusion*

If both the rule and the proposition are asserted to be true, then the conclusion must also be true: it is a necessary consequence of them. Here, knowledge of the nature of materials and of the particular artefact in question allows an inference to be made about the artefact, in this case a prediction about its physical properties.

Though it is not the only form of deductive inference, it will prove useful for the discussion later in this chapter to consider the abstract form of the syllogism given above:

$p \rightarrow q$	*rule ("p implies q")*
p	*proposition ("p is true")*
———	*(therefore)*
q	*conclusion ("q is true")*

where p and q are propositions. From the knowledge that the truth of q necessarily follows from that of p, along with the assertion that p is true, then q also is true.

The strength of deduction lies in its *truth preserving* nature - true premises entail a true conclusion. However, this is also a weakness, in a sense, in that it can be argued that a deductive process never *creates* knowledge, but rather it reveals facts that are already implicit in the premises. Hence, deduction may best be viewed as an analytical process, revealing truths implicit in that which is already known.

2.1.1 Deduction and Computers

In computational terms, deduction is relatively well understood, and automated theorem provers exist that can decide if a given proposition follows from what is already known (the well-known PROLOG programming language may be seen as one such system). In fact, all computer programs may be viewed as operating deductively, applying their code (rules) to data (propositions) to compute their conclusions.

2.2 Induction

Induction is the name given to the process by which an inference is made from the particular to the general. That is, given a number of empirical data, each describing a particular event of a common type, an inductive inference asserts that some property of the data is true for all events of that type.

In the context of this work, induction might allow a designer to learn from experiences of successfully designed artefacts, by, for example, relating elements of the artefact to its observed performance. For example:

artefact 1 is made of aluminium AND artefact 1 does not rust
artefact 2 is made of aluminium AND artefact 2 does not rust
...
artefact *n* is made of aluminium AND artefact *n* does not rust *premises*

IF artefact *x* is made of aluminium
THEN artefact *x* does not rust *rule*

In other words, the premises, a series of observations about the characteristics of artefacts, allow the induction of a generalised rule. This rule could then be used deductively to predict the properties of untested artefacts. It should be evident, then, that induction allows learning to occur, moving from experiences of specific instances to general 'rules of thumb' that permit appropriate responses to similar events in the future.

However, unlike deduction, the truth of the conclusion does not follow from the truth of the premises: it is supported, but not entailed, by them. If the next aluminium artefact to be observed happens to have rusted, then the original rule is false, in spite of the truth of the original observations.

A more formal principle of induction can be stated as follows (Russell, 1912):

> *"(a). When a thing of a certain sort A has been found to be associated with a thing of a certain other sort B, and has never been found dissociated from a thing of the sort B, the greater the number of cases in which A and B have been associated, the greater is the probability that they will be associated in a fresh case in which one of them is known to be present;*
>
> *"(b). Under the same circumstances, a sufficient number of cases of association will make the probability of a fresh association nearly a certainty, and will make it approach certainty without limit."*

From this, the degree of certainty attached to the conclusion increases with the number of corroborative examples seen; however, a single counterexample is sufficient to disprove the conclusion. Hence, the conclusion can never be considered to be indubitably true, it has merely a greater or lesser probability of being true, according to the amount of evidence to support it.

2.2.1 The Problems with Induction

There are, though, a number of problems that arise when considering induction:

- How many examples are sufficient to permit a useful conclusion to be drawn? Five unrusted aluminium artefacts? Ten? One hundred? More?
- Often, the observations will permit a very great number of inductions to be made, of which some are useful and others less so. For example, from the observed premises given in the above example, the rule:

  IF artefact *x* does not rust THEN artefact *x* is made of aluminium

  could equally well be inferred. Now, this might be a justified induction, given the premises, but is one that few people would make. Some manner of controlling the conclusions that are drawn must be applied. In this case, the additional 'common sense' knowledge that an artefact's properties arise from the nature of its material, and not vice versa, must be invoked. The mere *association* of thing A with thing B does not, in itself, seem a sufficient basis for making a useful induction. More generally, this difficulty is termed *Goodman's problem* or *paradox* (Goodman, 1983).
- The principle of induction seems to be circular. It implicitly contains an assumption of the uniformity of nature: that the future will, in all appropriate aspects, continue to resemble the past. But such an assertion can itself only be based on induction, and so calls into question the validity of any inferences based on this assumption. This is the notorious *problem of induction*, first raised by David Hume in his *An Enquiry Concerning Human Understanding* of 1748, reprinted in (Hume, 1975).

In spite of these difficulties, induction is a powerful, pragmatic reasoning tool. In contrast to deduction, induction *does* permit the production of new knowledge beyond that directly implied by the premises. This is in the form of generalised 'rules', which can be used to predict the outcome of future events. The drawback is that the use of an induction in reasoning introduces an element of uncertainty into the process, since it can never be conclusively proved, and remains ever-susceptible to disproof.

2.2.2 Induction on Computer

In terms of implementing induction on computers, recent work in the *machine learning* field of AI has resulted in the postulation of a number of different models of automated inductive learners. In different ways, these attempt to overcome some of the problems raised above to produce models of inference that share certain characteristics with typical human induction from data. Inductive machine learning, and its use in design problems, will be discussed in greater detail in later chapters.

2.3 Abduction

The third form of reasoning considered here is abduction, or *inference to the best explanation* (Harman, 1965). Abduction is a form of inference that goes from data describing something to a hypothesis that best explains or accounts for the data (Josephson and Josephson, 1994). The term 'abduction' was introduced by the philosopher and logician Charles Sanders Peirce, who was the first to recognise it as a distinct form of reasoning (Peirce, 1940).

In engineering contexts, this process can lead to reasoning such as the following:

IF an artefact is made of aluminium	
THEN the artefact does not rust	*rule*
artefact x does not rust	*proposition*
———	
artefact x is made of aluminium	*conclusion*

Hence, this combination of causal rule and proposition allows an inference to be made. However, and as is the case with inductive reasoning, the truth of this inference is not guaranteed by the truth of the premises – the artefact may be made of some alternative non-corroding material. So, if it is also known that the artefact does not conduct electricity, it might be better to abduce that the artefact is made of plastic. This is the most *plausible* explanation of the behaviour of the artefact, based on the available evidence – abduction is *plausible inference* (Charniak and McDermott, 1985). The strength of abductive reasoning lies in its generative capacity: if successful, it produces knowledge that is not implied by the premises. As might be expected, this capacity proves immensely useful in practice.

In more abstract terms, an abductive process of this type can be described as follows:

$p \rightarrow q$	*rule*
q	*proposition*
———	
p	*conclusion*

From this pattern, it can be seen that abduction has something of the nature of the 'reverse' of deduction: it involves reasoning 'backwards' from some consequent (effect) to an appropriate antecedent (cause). And just as deduction may be viewed as an analytical process, abduction, the 'reverse deduction', is synthetic – it produces knowledge (in the above example, the knowledge that the artefact is made of aluminium has been produced). While this is not a valid argument, on logical grounds, it might well be the case that 'unsound' arguments of precisely this sort are necessary for synthesising a design solution. The role of abductive reasoning in the generation of solutions to design problems is explored more fully later in this chapter.

More generally, Josephson and Josephson (1994) consider abduction in the following terms:

> *D* is a collection of data (facts, observations, givens).
> *H* explains *D* (would, if true, explain *D*).
> No other available hypothesis can explain *D* as well as *H* does.
> Therefore, *H* is probably true.

In other words, the body of data provides evidence for a hypothesis that satisfactorily explains or accounts for the data.[1] Josephson and Josephson identify the application of abductive reasoning in areas as diverse as medical diagnosis, scientific theory formulation and legal adjudication. In fact, it is a powerful everyday reasoning process, one which allows useful inferences to be made in situations where only partial or uncertain information is available. As such, it can be seen to encompass a great deal of the reasoning which humans apply to even the most mundane of tasks.

2.3.1 The Problems with Abduction

Abduction can generate knowledge that is not implicit in the premises. Again, this is at the expense of admitting the possibility of invalid inferences from true premises: abduction is

[1] In this respect, some have postulated that abduction is merely a special case of induction, and others (e.g., (Harman, 1965)) that induction is a special case of abduction, in which the task is to explain a number of observations. Whatever the merits of these opinions, it is useful to preserve a distinction here.

fallible reasoning. An abductive inference may be invalid for any of the following reasons (Josephson and Josephson, 1994):

- The data are wrong, or some are missing.
- Plausible hypotheses are unknown (due to incomplete knowledge).
- Hypotheses are incorrectly judged to be implausible (due to incorrect knowledge, or missing data).
- Hypotheses are incorrectly thought not to explain data (due to incorrect knowledge).
- A hypothesis is incorrectly thought to explain data (due to incorrect knowledge).
- The abductive inference is incorrectly thought to be better than it is (due to incorrect knowledge or missing data).
- The true inference is underrated (due to incorrect knowledge or missing data).

Beyond the failure of a particular abductive episode, there are a number of more general difficulties that arise when considering abduction:

- What triggers abductive reasoning? *Ad hoc* explanations are not formed to account for each piece of data that is observed. (Peirce suggested that the process is instigated when a *surprising* fact was observed, which intellectual disquiet would insist be explained; a more pragmatic view might be that an explanation is sought whenever the *need* to explain something arises.)
- How are explanations generated? There would seem to be a practically unlimited number of different hypotheses that can be generated to account for any given set of data. Most of these hypotheses will be highly unlikely, so some mechanism must control their postulation. However, without generating all possible hypotheses, it is impossible to guarantee that the best explanation will be considered. In any case, it is extremely unlikely that all possible hypotheses would be available for consideration, many being simply unknown to the abductor. Hence, there is a distinct element of subjectivity in the explanations generated. Furthermore, the premises may only be satisfactorily explained by some combination of 'atomic' hypotheses. For example, a doctor might ascribe a patient's symptoms to the presence of two independent diseases.

  A related issue is the need to control the *type* of cause that might qualify as a valid explanation. For example, an illness might be explained as being due to the presence of a particular infection in the patient's body, or to the poor quality of the available drinking water, or to the low socio-economic standing of the patient's country. Each of

these explanations can be seen as a valid cause of the illness. However, their validity as successful abductions must be viewed in the context of the need for the explanation: the first explanation is useful to the doctor, the second to the epidemiologist, the third to the social commentator. Josephson and Josephson summarise this point when they remark:

> *"the things that will satisfy us as accounting for [a finding]* f *will depend on why we are trying to account for* f*; but the only things that will count as candidates are parts of what we take to be the causal ancestry of* f*."*

Bylander *et al.* (1991) present the result that, even in the simplified domain in which they investigate, finding the best abductive hypothesis is, in general, an *NP-complete* problem. Informally, this means that the computational time required to guarantee a correct solution is found varies exponentially with the size of the problem; an incremental increase in the size of the problem results in an incommensurate increase in the amount of time necessary.

- Upon what criteria is the judgement of the *best* (*most plausible*) available explanation based? Josephson and Josephson highlight a number of considerations when making this decision:
  - How decisively an explanation surpasses the alternatives.
  - How good the explanation is by itself, independently of considering the alternatives (i.e. likelihood is not just relative: a poor explanation, even though it be the best available, should not be accepted in certain situations).
  - Judgements of the reliability of the data.
  - How much confidence there is that all plausible explanations have been considered (how thorough was the search for alternative explanations?).
  - Pragmatic considerations, including the cost of being wrong and the benefits of being right.
  - How strong the need is to come to a conclusion at all - it may be better to defer the decision and seek further evidence.

So, it would seem that even ostensibly simple explanatory tasks can belie quite complex reasoning, relying on astute judgement and demanding difficult decisions be made.

2.3.2 Abduction on Computer

So, the strength of abductive reasoning lies in its generative capacity: if successful, it produces knowledge that is not implied by the premises. As might be expected, this capacity proves immensely useful in practice. Unfortunately, as the above list of difficulties should indicate, abductive reasoning is poorly understood, especially when compared to deduction.

Since abduction may be viewed as the 'reverse' of deduction, it might be supposed that it is possible to simply run a deductive system 'backwards', so to speak. To try to illustrate this, consider a system comprising a rule of the form:

$$p \rightarrow q$$

Operating deductively, given the proposition p, this rule allows q to be inferred. Conversely, given q, the rule would also seem to allow the conclusion that p is the cause of q to be made. While this might indeed be the case, difficulties arise if the system also contains the rules:

$$r \rightarrow q \qquad \text{and} \qquad s \rightarrow q$$

These present no problem when reasoning deductively, but now the system contains three possible causes of q, namely p, r or s. An essential element of abduction lies in the choice of the best (most plausible) amongst a number of competing explanations and in a deductive system there is no guidance for making this choice.

So, some alternative method of performing abductive tasks is necessary. Indeed, the recognition of the power of abduction, and its naturalness as an everyday form of human reasoning, has led to a certain amount of AI research being devoted to it, and the proposal of a number of computable models of abductive reasoning. One example is the *RED* system (Smith *et al.*, 1985) for red-cell antibody identification, which was later developed into the more general *Peirce* system (Punch *et al.*, 1990). However, these systems demand complete knowledge of the associations between causes and effects, using this to search for good explanations (typically, the explanation of a composite effect which involves the postulation of the least number of causes is considered to be the best). This complete causal knowledge is unlikely to be available for the majority of complex abductive tasks, and, in general, a simple search for minimal sets of hypotheses is unlikely to alight on a good solution.

Accordingly, many applications of abductive reasoning make use of *heuristic* knowledge, rules of thumb that, while unable to ensure a solution, have nonetheless been found useful for making progress towards solving the task. Accurate heuristic knowledge can have the effect of making good (i.e., useful) abductive inferences, inferences that turn out to be true a high proportion of the time. Using this knowledge, abduction can then be viewed as

'probable deduction' (this idea will be explained more fully later in this chapter). The following chapter discusses heuristic knowledge in greater depth, and provides some examples of *knowledge-based systems*, systems that are able to reason with these heuristics, and, as such, can be seen to perform abductive reasoning.

First, however, consideration will be given to the roles that each of the three modes of reasoning, deduction, induction and abduction, play in solving design problems.

2.4 A Model of Rational Design

From the foregoing discussion, it is tempting to postulate a logic of design based on deductive reasoning, so as to exploit its truth-preserving nature – the process would be guaranteed to result in a 'true' design solution. So, given:

> S, a design specification
> K_D, knowledge of how to design in this domain
> ———
> Deduce D, a design solution

In other words, from the design specification, S, and the appropriate knowledge of how to design, K_D, a design solution, D, necessarily follows (assuming that S and K_D are consistent).[2] However, deduction is only applicable in situations in which complete, true premises are available. For any realistic design problem, this would be an extremely unlikely state of affairs. As Takeda (1994) comments, "although solutions and knowledge are always incomplete in design, [this deductive approach] requires solid and absolute knowledge and solutions". Designers do not have explicit prior knowledge of how to translate all possible specifications into design solutions; rather, they use their generalised heuristics to devise solutions to novel problems. Hence, this approach of 'design by deduction' cannot represent a plausible general model of design.

Consider, then, the following. Given:

> D, a design solution
> K_T, a theory of the domain
> ———
> Deduce S, a specification of the design

[2] Throughout this section, the exact nature of each of the quantities is intentionally left vague. This should not harm the argument, while preserving its generality.

This states that from a particular design solution and a theory of the domain, K_T (which might include knowledge of components, their behaviours, connections etc.), it is possible to deduce a specification of the design solution, in terms of its function, performance, behaviour and so on. Now, this sort of reasoning would seem to be much more plausible, and better accords with the *analytical* nature of deduction – this is an analysis of the design solution. Designers *do* know about the constituents of their domains and are able to understand the rationale behind design solutions. (Note that there are a number of problems with this simple model; for example, a particularly original (but valid) design solution might well be inconsistent with the current theory of the domain, and its acceptance might necessitate a revision of the theory. However, in general, this model is adequate for the consideration of design in this research.)

The problem of how to effect the *synthesis* of design solutions remains, though. In its analysis, a successful design solution can be seen, in a sense, as the *cause* of the specification, inasmuch as, if it were constructed, it would physically bring about the specification. Now, in the conceptual design task, the starting point of the process is a statement of the specification from which a solution is to be generated. In other words, an effective cause of the specification is sought: the process is the 'reverse' of the deductive analysis. This accords with an abductive model of reasoning: the task is to 'best explain' the specification by inferring a good design solution that will achieve it.

Before discussing a model of design reasoning which makes use of these ideas, it should be noted that there is a subtle difference between this use of abductive reasoning and its more general applications, as introduced above. In design, the specification 'effect' remains, as yet, a hypothetical one – it has not actually been observed – and the solution, once constructed, would serve to bring about the desired effect. The relationship is a subjunctive one - if this specification *had been* observed, this design scheme would have caused it. In design synthesis, the causes and effects reside in the mind of the designer. However, whilst this may shift the elements under consideration from the physical sphere to the mental, it should not affect the nature of the abductive task itself.

2.4.1 The March Model of Design

The abductive nature of design synthesis has been recognised by a number of authors (e.g., Coyne (1988), Coyne *et al.* (1990), Cross (1994), Takeda (1994)). March (1976) specifies a model of design using this idea (Figure 3). In contrast to the model of design of Pahl and Beitz presented in the first chapter, this model is more one of the *designer*. As such, it is an

account of how the design task is performed in terms of the reasoning employed to bring about the synthesis and analysis required at each stage.

In this model, design is a cyclic procedure, consisting of the following steps, generally proceeding in a clockwise direction, with reference to the figure:

1. By a process of production (March's "more telling and natural" term for abduction), the combination of (specification) data and models of the domain are translated into a proposed design solution.
2. Using the domain theory, the performance of this solution can be deductively predicted. If it meets the specification, then the process has been successfully completed.
3. The characteristics of the solution so derived, and the solution itself, permit an inductive modification of the suppositions underpinning the domain theory. If the solution has failed, for whatever reason, this revision of the theory allows the designer to return to step one and propose a better solution, and so on.

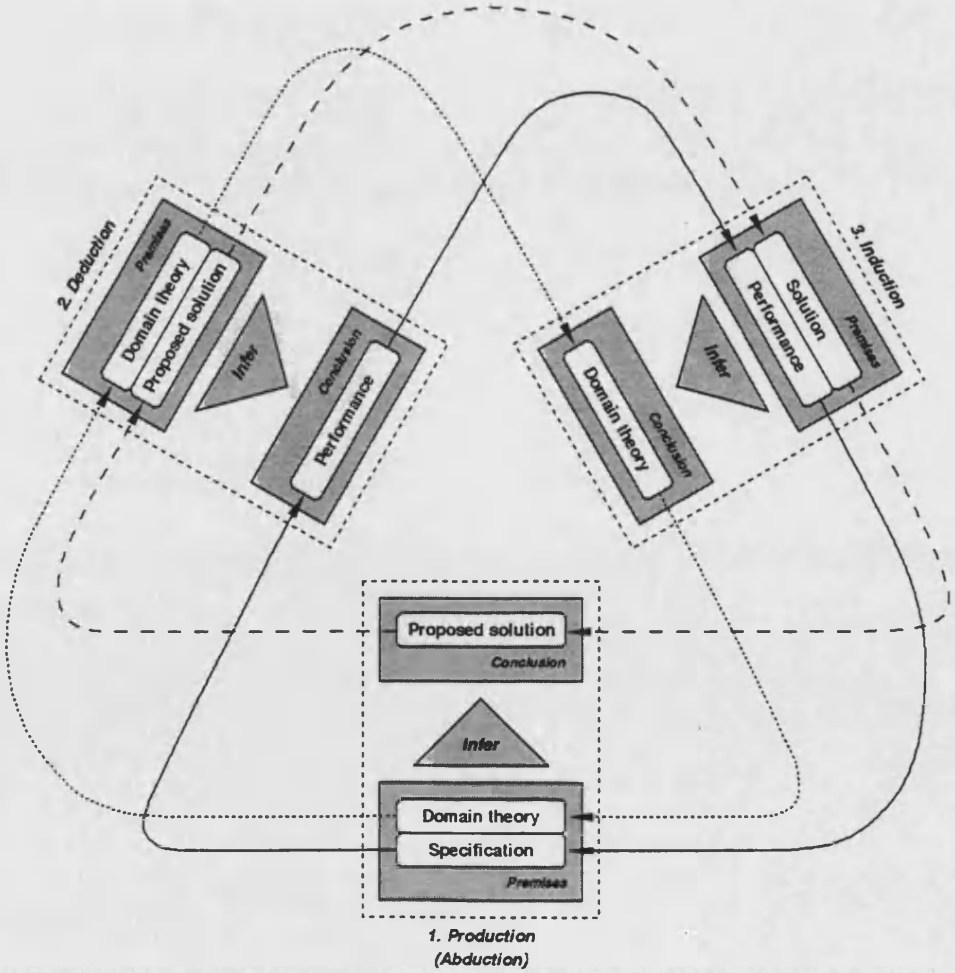


Figure 3. A model of design reasoning (after March (1976)).

The introduction of the third reasoning element, the process of induction, explains how the knowledge required to perform the abductive synthesis can be constructed. The deductive analysis of the proposed solution highlights areas in which the solution succeeds in meeting its specification, and those in which it fails. This information can be used to modify the knowledge used to bring about the abductive inference of solutions. Hence, by this process of adaptation, the solution to a particular design episode can gradually be evolved into a more correct form, and also, the design knowledge can be improved to increase the effectiveness and efficiency of future design episodes.

2.5 Limitations of the March Model

Whilst March's model may seem to present a plausible description of the logic applied by the designer (and it is one that has been advocated by a number of researchers), Zeng and Cheng (1991) and Roozenburg (1993) point out some difficulties with this model. Principally, these concern the role assigned to abduction. Abduction reasoning, as it has been discussed thus far, has been *explanatory* in nature: abduction is used to explain some effect by appealing to some cause likely to have brought it about in the current circumstances. However, this assumes that possible causes, and their associations with effects, are already known – and, typically, this will not be the case in conceptual design synthesis problems.

To illustrate this, consider the following example:

IF an artefact has form *f* and is used in manner *u*
THEN the artefact will meet specification *s* — *rule*
artefact *x* must meet specification *s* — *proposition*

---

artefact *x* should have form *f* and be used in manner *u* — *conclusion*

which conforms to the description of abductive design reasoning, as given above; from the designer's knowledge, and the current specification, the design of the artefact is inferred. However, this can only represent a valid model of reasoning if the form and manner of usage of an artefact that can achieve the specification are known *a priori*. Especially in original design (in the sense in which the term is used by Pahl and Beitz), these will not always be known beforehand – and it is precisely the task of the designer to devise the artefact's form and use in such cases. As Zeng and Cheng state:

> *"March has taken abductive reasoning as the logic of design. As abductive reasoning requires, a particular theory [i.e., rule] should hold in the*

> *inference.... March's proposition implicitly assumes that the form of the design is a priori determined, which is not the case in design. Indeed, design is a process to simultaneously produce both the artifact and its behaviour system which is here the major premise of the logic. As a result, the theory dominating the inference isn't known."*

Hence, the model of the reasoning in original design would seem to be something similar to:

artefact *x* must meet specification *s*	*proposition*
———	*(therefore)*
IF an artefact has form *f* and is used in manner *u*	
THEN the artefact will meet specification *s*	*rule conclusion*
artefact *x* should have form *f* and be used in manner *u*	*design conclusion*

or, more abstractly:

q	*proposition*
———	
$p \rightarrow q$	*rule conclusion*
p	*conclusion*

So, from a specification proposition alone, both a causal rule and, from this, a design are inferred. Roozenburg, following Habermas (1968), terms this mode of reasoning *innovative* abduction, to distinguish it from its *explanatory* counterpart.

This description of design synthesis would appear to present something of a paradox. It suggests that the knowledge (rule) used to bring about the design is not known before the design process commences. Furthermore, nor are some of the elements used to compose this knowledge – namely, the artefact's form and usage. Nonetheless, it is the role of the designer to generate these descriptions of form and usage by applying the knowledge – and, at the same time, to generate the knowledge using the descriptions of form and usage! Obviously, this is a circular argument.

Zeng and Cheng suggest that the circle can be broken using a recursive process of:

> *"...inference of a case [i.e., a solution conclusion] and a partial rule from a result [i.e., a specification proposition], which reflects the designer's presumption that a certain form might exist to satisfy the functions required."*

In other words, the fallacy of this reasoning lies in viewing design synthesis as a single operation, when, in fact, it proceeds through a number of iterations, all the while building

upon and developing the knowledge of the designer. This explanation accords well with the idea of design as an iterative process as described by Pahl and Beitz and many others. An alternative approach might be to introduce from a second domain some form and usage that produces analogous behaviour or functionality in that domain, hypothesising that (once translated appropriately) they will have a similar effect in the working domain.

Whatever the approach, performing wholly original design appears dreadfully complicated. Fortunately, the difficulties are eased somewhat in the case of the research reported here, since the design task in question is one of configuration design. So, whilst the system of components that will achieve the required specification might be unknown at the outset of the design process, the components themselves, and their behaviours, conventional usages and the manners in which they are connected *are* known. In other words, the basic solution principles, lacking in original design tasks, may be considered available to the designer. Knowledge of these can be used to generate a system of components, the combination of which, it is hypothesised, will meet the combination of behaviours or functions indicated by the design specification.

So, the configuration design problem might be viewed in terms of a series of individual explanatory abductive processes:

IF a system includes component *c*, used in manner *u*
THEN the system can provide element *s* of specification *rule*
system *x* must provide element *s* of specification
proposition

---

system *x* should include component *c*, used in manner *u* *conclusion*

and then hypothesising that the combination of components selected to meet the individual elements of the specification will together form a system that meets the whole specification. Obviously, there are potential flaws in this reasoning – the component may not have the desired effect in the current circumstances, the combination of components may interact and as a result fail to meet the specification, and so on – but abduction is fallible reasoning, and designs can fail. Successful design synthesis is dependent on the experience of the designer, knowing which components, used in which ways and combinations, are most likely to meet the specification in the given context. The aim of the research described in this thesis is the emulation of this experience-based reasoning on computer.

2.6 The Implementation of Design Reasoning on Computer

Regardless of whether it conforms to notions of explanatory or of innovative abduction, Roozenburg is pessimistic about the chances of successfully automating design at this time. Quoting Coyne (1988):

> *"Pending the development of a radically new type of computer it is safe to say that that which cannot be formalised in logic cannot be modelled in a computer system."* (p. 11)

he concludes that, since computers apply deductive logic, abduction (and, consequently, design synthesis) cannot be automated using current technology.

However, this seems to be confusing theory and practice, confusing an implementation and its effect. Certainly, a digital computer does act in a deductive manner, but, where intelligent systems are concerned, this simply describes the manner of the application of knowledge during a problem-solving episode, not the nature of the reasoning that results from this application. As will be seen in the following chapter, the use of *heuristics*, knowledge which is not absolute, and from which the truth of the inferences made cannot be guaranteed, can realise the essential nature of abductive reasoning. Typically, for configuration design, these heuristics have something of the form of the 'inverse' of the rule used in the formal description of explanatory abduction. As an illustration, the rule:

> IF a system includes component *c*, used in manner *u*
>
> THEN the system can provide element *s* of the specification

may be 'inverted' to give the heuristic:

> IF the system must provide element *s* of the specification
>
> THEN the system may require component *c*, used in manner *u*

As seen above, with the attempt to run a deductive system 'backwards', this rule is unlikely to be useful as it stands - there might be a number of ways of providing element *s* of the specification and no information to allow a choice to be made amongst these alternatives. However, some basis for making this decision can be provided, by the addition of contextual information from experience. Accordingly:

> IF the system must provide element *s* of the specification under conditions *z*
>
> THEN a system requires component *c*, used in manner *u*

Now, a specification demanding this particular element under these, or similar, conditions allows this heuristic to be judged to be the most appropriate, and as a result, the designer can

assume it to be true. In other words, the component *will* satisfy the element of the specification in this case. Consequently, it can be used *deductively* to add components to the design to meet that element of the specification. This is still fallible reasoning – the designer's judgement may be poor or knowledge lacking, and the rule may not, in fact, be valid in the current circumstances. Moreover, heuristic knowledge must remain open to revision and correction in response to new information and, as a result, cannot be regarded as 'complete'. Hence, the conclusions that are drawn may be considered abductive in nature, having associated some degree of doubt. Nevertheless, this provides a mechanism ('probable deduction') by which progress can be made towards the synthesis of designs. As would be expected, designers having more extensive and accurate heuristic knowledge would seem to be more likely to arrive at a correct solution.

Furthermore, as Roozenburg overlooks, this would appear to be a plausible description of the sort of approach adopted by human designers, and not merely an expedient adopted to enable digital computers to perform synthetic design tasks.

2.7 Summary

As treated here, the reasoning that a configuration designer applies to synthesise designs conforms to instances of explanatory abductive inference. This has a number of implications, for both human and computer-based design. Principal amongst these is the realisation that, in general, synthesis results from a process that, logically speaking, is not sound. Abduction is inference to the *most likely* hypothesis: there can be no guarantee of its validity. However, the acceptance of some degree of uncertainty into the process seems to be, in some respects, a trade-off against the gains to be had from the useful production of new ideas through this form of reasoning.

It follows from this model of design that the more successful designers are those who are able to generate 'most likely' hypotheses (i.e., design solutions) that are found, in practice, to correspond to demonstrable truths (i.e., artefacts or systems that meet to the specification) a greater proportion of the time. To achieve this sort of successful abductive inference, there would seem to be two overriding considerations. First, since any judgement of this sort is based upon incoming information (in this case, a design specification), a competent designer must have a good notion of what information is relevant (and, equally, what is not). Secondly, the designer must have the heuristic knowledge that allows the proposal of good hypotheses in response to this information. This knowledge would seem to be the result of the many years of experience that can be necessary to achieve expertise in a domain. The

research described here is an investigation into how this experience can be provided to a computer system, and then exploited to generate design solutions in the future.

However, a further implication of the reliance on abductive reasoning for design synthesis should be noted: it would seem to suggest that, in general, there can be no such thing as the 'ideal' designer, who (or which) is guaranteed to devise a correct design solution in response to every possible design problem.

3 Knowledge, Design and Intelligent Systems

Any intelligent behaviour, such as design, is invariably bound to the idea of knowledge: that designers have to know how to perform their task seems a trivial observation to make. Nonetheless, when considering the automation of this behaviour on computer, it becomes necessary to think in greater detail about what this means. This chapter is devoted to a discussion of knowledge and of some of the ways in which knowledge has been stored and applied in computer systems. Naturally, the emphasis here is placed upon design knowledge and intelligent design systems.

This chapter begins with a brief introduction to the nature of knowledge and how it is considered by AI practitioners. This leads into a discussion of knowledge *representation* – the ways in which knowledge may be expressed, stored and used on computer.

Following this, consideration is given to the nature of knowledge in design, and a model of knowledge types, and their relationships, in design systems is presented. As suggested in the previous chapter, the ability to make design decisions, to reason abductively, requires the application of *heuristic* knowledge. Accordingly, any automated design system must incorporate this sort of knowledge in some manner. Most previous attempts to build such design systems have relied on *knowledge engineering* (Michie, 1973) to capture these heuristics. Knowledge engineering techniques are used to try to acquire knowledge directly from human experts for the purpose of building intelligent systems. These techniques are discussed in this chapter, as are the difficulties that surround the acquisition of heuristics in this manner.

Previous attempts to automate design may usefully be divided into *Knowledge-Based Systems* approaches and *Case-Based Reasoning* approaches. Within the former, the heuristic knowledge is explicitly represented and used to generate new designs. On the other hand, in a Case-Based Reasoning system, at least some of the heuristics in the system are implicit, stored in the form of examples of previous design episodes. When faced with a new design problem, an appeal is made to the knowledge contained in these examples to suggest a solution. Both of these approaches will be described in detail in this chapter, along with a review of some of the automated design systems developed using these ideas.

3.1 Knowledge and AI

For the purposes of this work, it is helpful to take a practical view of knowledge – knowledge is that which allows an entity to act intelligently, to respond usefully to the environment and situations in which it finds itself (Newell, 1982). As such, knowledge is wholly subjective – it is not and cannot be independent of the entity. Any attempt to convey knowledge to a second party can only be done by providing *information*, expressed in terms of explicit symbols. The second party may, or may not, be able to 'internalise' this information as knowledge – in other words, store it so that it can be used in future action.

AI is expressly concerned with making 'knowledge' in some way explicit, so that it can be incorporated within computer systems. Hence, it might be better to consider this knowledge to be the information necessary to support intelligent reasoning (Partridge, 1996). As Hart (1988) remarks, "knowledge, as stored and processed by computers, is a high level of information." This information can be inspected at any time; usually it will be expressed in the form of symbols. When this information is put to use and interpreted by some mechanism, hopefully something like intelligent behaviour will be evident. Throughout this and subsequent chapters, then, the term 'knowledge' is used as shorthand for the 'information stored on computer, which, when manipulated by appropriate mechanisms, results in apparently intelligent behaviour'.

So then, knowledge – human or computer - rests in an entity's memory, until such time as it is used, when it is retrieved and applied to the current situation. This raises the issue of the *representation* of knowledge. However, before discussing knowledge representation, consideration will first be briefly given to the distinction between *declarative* and *procedural* knowledge.

3.1.1 Declarative and Procedural Knowledge

Traditionally, an epistemological distinction has been made between declarative knowledge (that is, factual knowledge, knowledge *that* something is so) and procedural knowledge – knowing *how* to do something (Stillings *et al.*, 1987). In any sort of intelligent problem solving, knowledge of both forms appears to be necessary. In design systems, declarative knowledge would seem to be present in the form of knowledge of the entities of the design problem – for example, the components that may be used to construct a configuration. On the other hand, procedural knowledge would seem to describe how the design process is actually performed. As shall be seen, it is this procedural 'how to' knowledge that is particularly difficult to acquire for computer systems. This is perhaps unsurprising, since

procedural knowledge, by its very nature, would seem to be more difficult to express explicitly as information than would declarative knowledge.

3.1.2 Knowledge Representation

The term 'knowledge representation' is used to describe the manner in which knowledge is organised and manipulated in the system's memory. One of the tenets of AI is the importance of knowledge representation in producing intelligent behaviour – in other words, such behaviour is not merely a factor of the content of the knowledge.

Any intelligent system acts within and upon its environment. Accordingly, the system must have some sort of internal representation of its environment, a mapping onto the actual declarative facts of the world. In this way, a representation is a stylised version of the world (Charniak and McDermott, 1985). The same facts may be represented in different ways for different purposes, but it is important that the internal representation of this knowledge in some way captures the essential structure and relationships of the world. Through this structuring some semantics can be assigned to the symbols with which AI systems (usually) operate. In addition, the procedural knowledge that manipulates these facts must also be represented in an appropriate fashion.

Representations can be of greater or lesser complexity, depending on the complexity of the task they describe. To be useful, a representation must offer more than merely the static storage of facts: it must provide the appropriate mechanisms for dynamically accessing and manipulating its content, as and when the task demands.

When coming to model intelligent behaviour, then, it is necessary to suggest answers to the following questions (Stillings *et al.*, 1987):

1. What is the knowledge involved in the performance of the task, its types, structure and organisation?
2. How is this knowledge to be represented in the system?
3. Does the chosen representation reflect the natural structure of the task knowledge? Is it adequate for the task? How does it bias the knowledge content?
4. How is the knowledge to be acquired and/or revised?

Knowledge Representation Issues

To be useful, a knowledge representation must possess the following qualities (Rich and Knight, 1991):

- *Representational adequacy* – the ability to represent all of the kinds of knowledge that are needed in the domain for the task in hand.
- *Inferential adequacy and efficiency* – an appropriateness for making the sort of inferences necessary to perform the task.
- *Acquisitional efficiency* – the facility for incorporating new knowledge with ease.

Unfortunately, whilst a number of representational schemes have been proposed, no one scheme embodies all of these qualities for all tasks (which suggests that different representations are necessary when performing different tasks). The following section briefly discusses some of the representations that have been proposed in AI: for a more complete treatment see (Rich and Knight, 1991).

Knowledge Representation Techniques

There have been a number of alternative computer-based representations suggested by the AI community. In general, these are suggested by psychological models of knowledge structure, and supported by some empirical evidence. Example techniques include:

- *Attribute-value pairs* – a simple, frequently used representation: some domain entity is represented as an attribute symbol, which, according to the state of the entity, is assigned a particular value. (This form of representation is that commonly used with machine learning algorithms, as will be seen in the following chapter.)
- *Frames* (Minsky, 1975) – a frame is a collection of attributes describing some generic entity. A particular instance of the entity is represented as a series of values of these attributes. A value can be 'inherited' from the generalised frame description, or it can itself be a (pointer to a) frame, so establishing a network of relationships. Similar representations can be found in object-oriented programming languages.
- *Semantic networks* (Quillian, 1968) - the nodes of the network are symbols describing the known entities, with the arcs of the network representing relationships between them.
- *Predicate logic* – knowledge is represented as well-formed formulae in (usually) first order predicate logic.
- *Rules* – knowledge can be expressed in an *if-then* form: *if X then Y*, where X, the antecedent, is some context or data, which, if it is true, allows, Y, the conclusion, to be inferred. If Y is some action to be performed, the rule is being used to express procedural knowledge, stating an appropriate step to take in response to situation X.

- *Connectionist representations* - inspired by models of the brain, a connectionist representation (also called an *artificial neural network*) expresses knowledge as the state of a network of connected mathematical nodes. This differs from the other representations in that concepts can be expressed without the need for explicit symbols denoting them.

From this, it can be seen that a particular representation will be more suited to expressing certain types of knowledge; for instance, rules seem suited to representing procedural knowledge, while attribute-value pairs seem more appropriate for describing declarative knowledge.

Chapter 5 describes the manner devised during this research for representing fluid power systems design specifications and solutions using attribute-value pairs. This provides an internal description of the external elements of the task, and in such a way as, hopefully, to facilitate their manipulation during the design task. Later in this chapter, and in subsequent chapters, examples will be seen of the use of a number of different representations - including both rules and connectionist models - for expressing the sort of procedural design knowledge that reasons with these elements for design synthesis.

3.2 Knowledge and Design

Coyne *et al.* (1990) suggest that the range of design knowledge covers such things as laws, rules, and formulae pertaining to the behaviour of people, materials, objects and spaces. Added to this might be knowledge about the design problem itself, how it is described and what is available to solve it. Knowledge may be axiomatic and unequivocal, or casual and more heuristic in nature. They go on to say that:

> *"The knowledge with which we are concerned in design may also apply to different abstractions of design description. For example, we can talk of different control levels in design. There is knowledge about how components might fit together in a design. There is also knowledge about appropriate actions to perform in producing configurations, and knowledge about strategies. We must be able to represent and manipulate knowledge of this kind."* (p. 29)

They also stress the importance of knowing about generic designs, or design *prototypes*, which are, in general, designs from which other designs originate: a "prototype typifies, or exemplifies, a class of designs, and thus serves as a generic design." This definition is (intentionally) loose; prototypes could be explicit designs that can be re-used, or they may be

implicit, expressed, say, in the form of rules. As will be seen in chapter 5, a similar concept will prove useful in the representation and, later, for the design of fluid power systems.

Coyne *et al.* talk of different knowledge levels; one particular model of the different levels of knowledge that exist in a design synthesis system has proved useful in the course of the research reported here, and it will now be described.

3.3 A Model of Knowledge in a Design System

From the foregoing discussion, it should be clear that the key to successfully performing tasks that require intelligence is the proper application of correct knowledge. In AI, there have been a number of attempts made to classify, at a high-level, the types of knowledge that occur in problem solving, with the intention of modularising the expression and representation of each type. This section details a proposed model of the types of knowledge contained within a configuration design synthesis system, as devised by Schreiber *et al.* (1994) and Wielinga and Schreiber (1997).

In this model, the design knowledge (both declarative and procedural) is considered to be distributed across a number of different categories; the relationships between knowledge of these different types govern how this knowledge is used during design synthesis. Although they may not necessarily always be represented in the same fashion, and may differ in content according to the approach adopted, the assertion is that each knowledge type must be present in some form in any complete design system.

3.3.1 Domain Knowledge

This category contains knowledge of the entities that constitute the domain under consideration. In terms of configuration design, this might include knowledge of the physical elements (and their behaviours) which may constitute a solution, knowledge of the how these elements can be combined, knowledge of how groups of related elements (up to and including the system level) behave, component parameters, and so on. Presumably, some description of the elements of the design specification that the system 'understands' would need to be included. This category would seem to consist primarily of declarative knowledge – these elements correspond in some way to the external 'facts' of the design task.

3.3.2 Inference Knowledge

That is, knowledge of what conclusions (i.e., design decisions) may be made, and how to make them, given the current state of the process. In other words, how elements of the specification, along with the decisions already made, can be used to move closer to a solution. Hence, this type of knowledge is procedural, governing the transformations that can be made during the design process.

3.3.3 Strategic Knowledge

This is knowledge of how elements of inference knowledge can be arranged and controlled so as to provide a complete strategy for producing a design. This amounts to one or more high-level methodologies for controlling the process of transforming specifications into solutions. Again, this is procedural knowledge of the design process.

3.3.4 Working Knowledge

This is unique for each design episode and might contain the current design specification, design choices made, knowledge of the reasons for the modifications to a design, feedback from the customer about the application of the designed system, and so on. This category represents a 'pool' of knowledge about the current design process, from which elements may be retrieved when they are necessary for invoking or applying elements from the other categories of knowledge. This is primarily declarative knowledge, representing the facts of the current design problem.

Figure 4 shows these classes of knowledge - the more 'persistent', static knowledge can be seen in opposition to the less-enduring working knowledge, which exists only during the lifetime of the current design task, and which is subject to a greater amount of change. The categories are still quite loosely defined, and will vary from domain to domain, and may even vary within a domain when, say, different design strategies are applied, so this description cannot be considered as a generative definition for a design system. However, all these categories *must* be embodied and recognisable in some form within such a system, and as such, they offer some measure of the completeness of any proposed system.

One apparent omission from this model, though, is *common-sense knowledge*, that is, the knowledge thought to provide the foundations for all intelligent human behaviour. Implementing this sort of knowledge has proved to be an extremely difficult task for AI, with little success to date (the CYC project (Lenat, 1995) is one current long-term undertaking devoted to the 'codification' of common-sense).

Beyond recognising that this limitation is commonplace in AI systems, making them 'brittle' (that is, subject to failure) in the face of problems outside their narrow expertise, no further consideration will be given to this problem here.

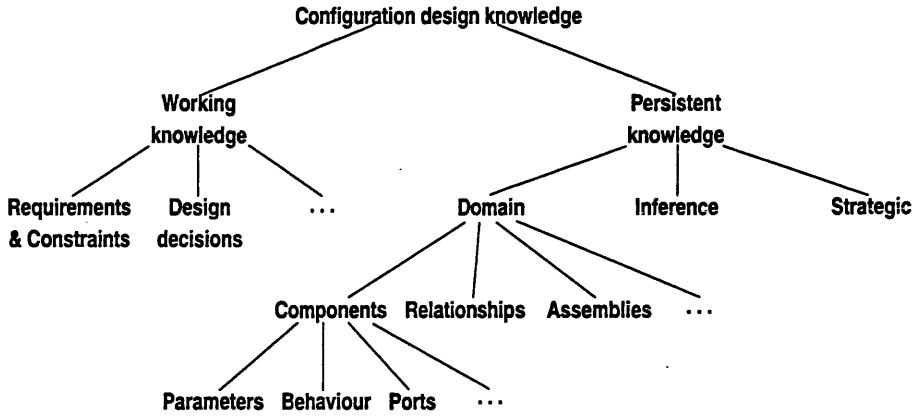


Figure 4. The hierarchy of design knowledge. From (Wielinga and Schreiber, 1997).

3.4 Expertise and Heuristics

Here, interest lies in a very particular form of intelligent behaviour – design, a task usually performed by experts. Hart (1988) remarks that:

> *"An expert is an expert* at something, *certainly not everything.... Experts have used their intelligence to develop a high level of expertise in a particular area, by training, education and learning from experience. In general, experts can solve problems with more versatility, efficiency, reliability and confidence than non-experts [can]."*

When trying to automate an expert behaviour, consideration must be given to the knowledge that experts apply, and the forms that it takes. In the analysis of the logic of design given in the previous chapter, the process of design synthesis was characterised as being reliant on abductive reasoning. As was described, this sort of reasoning is concerned with making judgements and decisions using incomplete or uncertain knowledge of the domain, based on experience of that domain.

This type of experience-based, uncertain knowledge is often termed *heuristic* knowledge. According to Fox (1996) heuristic knowledge is that which:

> *"...expresses rules of thumb, which are not guaranteed to be precise or correct, but [which are] nevertheless useful when...the field lacks a comprehensive body of theory and most practical knowledge is empirical."*

Clancey (1986) says:

> *"A* heuristic relation *is uncertain, based on assumptions of typicality, and is sometimes just a poorly understood correlation. A heuristic is often empirical, deriving from problem-solving experience; heuristics correspond to... 'rules of thumb,'....*
>
> *"Heuristics of this type reduce search by skipping over intermediate relations... These associations are usually uncertain because the intermediate relations may not hold in the specific case. Intermediate relations may be omitted because they are unobservable or poorly understood."*

In order to automate expert behaviour, then, it is necessary to incorporate this heuristic knowledge within a computer system. Most conventional approaches to constructing intelligent systems have relied on *knowledge engineering* to provide these heuristics.

3.4.1 Capturing Heuristics – Knowledge Engineering

Knowledge engineering is the extraction of useful knowledge (by *knowledge engineers*) from domain experts (Winston, 1993). A number of techniques have been developed or adopted to assist in this task. Not all these techniques are aimed at capturing the same sorts of knowledge, some aiming to access declarative knowledge, others procedural, and some a combination of forms. These techniques include:

- *interviews* – consists of asking the expert about the domain and the task in question. Interviews can be structured to varying degrees. The success of an interview session depends on the questions asked (the knowledge engineer must be familiar with the domain to a certain extent, and be able to control the session) and the ability of the expert to articulate his/her knowledge.
- *case study* – using examples of specific tasks to provide a context, the knowledge engineer asks the expert direct questions to obtain information. These cases may have been typical, difficult or memorable in some way.
- *protocol analysis* (Ericsson and Simon, 1984) – the expert solves a problem typical of the task while 'thinking aloud'. The intention is to capture the actions performed, the focus of attention at each stage and the overall strategy adopted. Its success is dependent on the ability of the expert to be able to do this fully and accurately. Furthermore, this act of self-reporting may actually interfere with the expert's performance of the task.

- *repertory grid analysis* (Kelly, 1955) – for each of a set of entities describing the domain or task, the expert is asked to assign a measure of the relative importance of a number of characteristics.
- *card sorting* – the expert orders cards representing domain entities according to some criterion; this is in order to get some idea how the expert classifies these entities and structures the domain.
- *laddering* – a hierarchical description of the domain entities is acquired by asking the expert a series of questions designed to elicit information about the elements above, below and at the same level of abstraction as a 'seed' entity.

Once the necessary information has been elicited from the expert, the knowledge engineer must decide how best to represent and encode it before it can be incorporated into the intelligent system. Sometimes, there is an intermediate stage – the knowledge engineer will express the knowledge in some formal manner which is comprehensible to the expert (as a decision tree, for example), and ask the expert to critique (and correct) it.

3.4.2 The Trouble with Knowledge Engineering

These techniques are, on the whole, extremely time-consuming, for both the expert and, especially, the knowledge engineer. If elicitation sessions are to be exploited to their greatest extent, a great deal of preparation must be made by the knowledge engineers. Typically, they will have to become familiar with the entities constituting the domain and, to a certain extent, with the task itself in order to control the acquisition process. However, as Winston (1993) observes, "knowledge engineering is an art, and some people become more skilled at it than do others." In other words, all the preparation in the world might not be enough to ensure a successful outcome.

The second essential ingredient in every case is a willing and able expert – and, for one reason or another, such an expert might not be available. Hart (1988) says that problems may occur if experts are:

> *"Inaccessible: [they] cannot be freed for periods of time because they are busy, abroad, or unavailable at the same time as other team members.*
>
> *"Not really an expert: it is not always easy to identify true experts, who must have knowledge relevant to the problem. Sometimes a team of experts is needed.*

> *"Inarticulate: completely unable to express and discuss the model of the knowledge.*
>
> *"Bored: less enthusiastic about the project or results to date, tired of answering the questions and without general commitment."*

Furthermore, as Hart goes on to say, problems can arise if experts lie, either to sabotage the project and preserve their status, or else because it "can be embarrassing for the experts to describe heuristics".

Once the interaction with the expert is completed, there still remains the laborious and difficult task of transcribing and analysing all that has been said and done, and then deciding how this may be best represented and encoded within the system. Quinlan (1986) remarks that:

> *"While the typical rate of knowledge elucidation by this method is a few rules per man day, an expert system for a complex task may require hundreds or even thousands of such rules. It is obvious that [the knowledge engineering] approach to knowledge acquisition cannot keep pace with the burgeoning demand for [intelligent] systems..."*

Another difficulty lies in determining the *completeness* of the knowledge gained in this fashion. For practical purposes, the knowledge may be considered complete if it is sufficient to describe and respond to the full range of problems encompassed by the task. Including knowledge of a very general nature can ensure some degree of completeness (but generalised knowledge is not always applicable to specific cases). The incompleteness of the knowledge may go unrecognised for some time, since it is usually impractical to test a system over all possible problems.

However, as Gillies (1996) observes, the problems with a knowledge engineering approach go beyond these:

> *"...in some cases, the experts may simply not know how they perform their skilled task, even though they perform it very well. In such cases, [knowledge engineering techniques] will have no success in producing a knowledge base for the computer to use." (p. 28)*

This knowledge of how to perform the task is precisely that heuristic knowledge that is required for design systems. This inability to articulate heuristics has also been documented by others (e.g., (Berry, 1986), (Hart, 1988)), and raises doubts about the accuracy of all heuristic knowledge acquired in this fashion: the heuristics that the experts supply may

reflect how they think the task *ought* to be performed, rather than how they actually do it. As Hart remarks:

> *"One important consequence of [the nature of expertise] is that it is not easy to get an expert to explain in verbal terms what he is doing and why he is doing it. [It] means that he cannot actually tell the full truth. The explanations will often be rationalisations and descriptions of rules and strategies for which he no longer has use. These rules and strategies...are not the natural way in which the expert thinks. This is demonstrated when experts reply with comments like:*
>
> *" 'You get a feel for it.'*
>
> *" 'I know that's right.'*
>
> *" 'You can sense it's different.' "*

The difficulties encountered are known collectively as the *knowledge bottleneck* in intelligent system development (Gillies, 1996). It should be evident that this bottleneck imposes serious limitations, both practical and theoretical, on the development of systems in this fashion.

So, given that the knowledge engineering approach is far from satisfactory, an alternative approach that is able to capture design heuristics in an accurate and complete fashion would be of immense benefit in the development of design systems. The idea that the heuristics are implicit in examples of design experts' work suggests that such an approach might be to exploit this source. This provides the motivation behind the research reported in this thesis.

3.5 Knowledge-Based Systems and Design

Conceptual design synthesis in complex domains relies on the application of heuristic knowledge. One approach to reasoning using heuristics that has arisen from AI work is through the use of *Knowledge-Based Systems* (KBS). The term 'Knowledge-Based System' is used in a number of different senses in the AI literature, but here it is used to refer to systems in which all the heuristics are expressed in an explicit form (for example, as a set of rules). (The next section describes an approach – *Case-Based Reasoning* - in which at least some of the heuristics remain implicit until they are needed during problem-solving.)

Typically, a KBS will be devoted to performing some particular task, rather than being a 'general' problem solver. It will include some model of the knowledge necessary to perform this task along with some generic reasoning mechanism which is able to use the knowledge. This is a major distinction between KBS and conventional computer programs - the

knowledge is kept separate from the methods of applying the knowledge. This means that the knowledge may be inspected, developed or modified, without the need to understand complex code, and could, in theory, be applied to other tasks.

Perhaps the most common way of expressing explicit heuristic knowledge is in the form of rules, and so, the following general discussion of KBSs concentrates on rule-based systems. However, it should be borne in mind that other representations of knowledge have been used in KBSs – examples of these will be seen later – and that the observations made apply equally well to these systems.

Typically, a rule takes the form:

if <some condition is true>
then <draw some conclusion>

The expert heuristics, stated in this form, together form the *knowledge base* of the system; the quality of the results generated by the system depends on the quality of this knowledge base. In addition to the knowledge base, the other principal components of a KBS are an *inference engine*, a *working memory* and a *user interface*. The inference engine (which is often task-independent) contains the mechanisms for selecting and applying elements of the knowledge base given the current state of the problem solving. The current state is described by the working memory, in terms of 'facts', or assertions that have been provided initially (through the user interface) or subsequently inferred (Figure 5).

In general, the inference engine will act in a deductive manner: if the working memory reveals that the conditions of a rule correspond with stored facts, then conclusion of this rule may be inferred and added to the memory. However, there may be a number of rules that seem equally applicable in the current state, in which case some mechanism must be invoked to determine the inference order. A typical mechanism might be to trigger first that rule which has the most specific true conditions, given the current state of the working memory.

To give a more concrete example, the *R1* system (McDermott, 1982) is a KBS devoted to a configuration design problem, in this case, the task of making up DEC VAX mainframe computers. This is one of the earliest design KBSs (and, indeed, one of the earliest and more successful KBSs in any field). The heuristic knowledge consists of rules relating the design specification to elements of the solution. Despite a number of problems (of which, more later), R1 was successful (and used commercially), but the relative simplicity of the chosen domain may have been a factor contributing to this success: component choice is assumed to be independent in nature and the number of arrangements is limited.

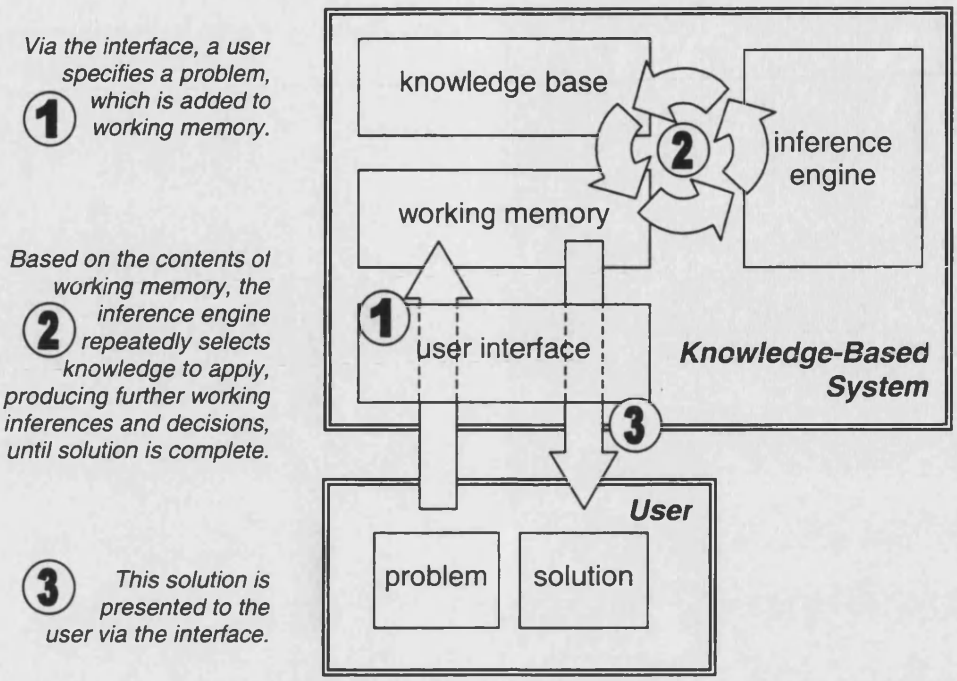


Figure 5. The typical contents and operation of a KBS.

A typical R1 rule looks something like (quoted by Rich and Knight (1991)):

```
if:
   the most current active context is distributing massbus devices, and
   there is a single-port disk drive that has not be assigned to a
   massbus, and
   there are no unassigned dual-port disk drives, and the number of
   devices that each massbus should support is known, and
   there is a massbus that has been assigned at least one disk drive and
   that should support additional disk drives,
   and the type of cable needed to connect the disk drive to the previous
   device on the massbus is known
then:
   assign the disk drive to the massbus.
```

It can be seen that, if a (relatively complicated) set of conditions is met, a conclusion can be reached, which, in this case, is to add a particular element to the configuration. Other, higher-level rules exist for controlling the overall strategy and focus of attention (for example, deciding at which point in the design process to consider the assignment of disk drives). Rules, then, represent procedural knowledge: in the above example, if the conditions are met, the design is moved one step nearer completion by the addition of another component. The complete procedural knowledge for the task resides in the combination of rules and inference engine. The declarative knowledge exists in the form of the entities to which the rules refer: for example 'disk drive' and 'massbus', symbols that represent physical components of the domain.

The principal appeal of using rules to represent heuristic knowledge is apparent from this example - in general, the rules are symbolic and qualitative in nature, and, as such, seem to embody a natural representation of a problem, and since they are expressed in a pseudo-natural language, they are readily comprehensible.

KBSs have been widely used, and for a number of different tasks (Hayes-Roth and Jacobstein, 1994). In practice, the need to maintain and develop the knowledge base presents a drawback to the use of KBSs. Often problems are found in the operation of a KBS, which are then 'repaired' by adding further heuristics or modifying existing ones to cope with error-prone situations. This, coupled with natural expansion of the scope of the system and the need to keep pace with evolving domains, can, if not done with care, produce serious, debilitating problems in the operation of the system. As the precise order of operation (in terms of the sequence in which rules are invoked) of a KBS depends on the particular problem under consideration, this can make the potential influence of new rules difficult to appreciate.

As an example of the difficulty of knowledge-based maintenance, in the (already complicated) rule quoted above from the R1 system, an incorrect inference might be rectified by the insertion of an additional clause in the conditions. However, this would also have the effect of making the rule more specific, and, depending on the inference mechanism in use, might mean that the rule is now being applied in situations where it previously was not – and so, this introduces the possibility of more errors in the system performance. Over the course of four years' development, the number of rules in the R1 system increased from less than eight hundred to more than three thousand (Bachant and McDermott, 1984). As a result of carrying out this development, Bachant and McDermott were in a position to observe that:

> *"...for the most part, adding a piece of knowledge involves some amount of creativity...by no means can this task be done without substantial amounts of problem solving.*
>
> *"It was clear before R1 was a year old that the incremental addition of knowledge resulted in a system with a significant amount of redundancy and a penchant for* ad hocery. *To the extent that adding knowledge to the system involves human intervention, this general lack of cleanliness and conciseness provides an obstacle to the system's further development."*

3.5.1 Knowledge-Based Systems in Engineering Design

KBSs operate using the sort of heuristic knowledge that would seem to be necessary for performing design synthesis, and, accordingly (notwithstanding the problems mentioned above), KBSs have been applied, and with some success, to performing engineering design synthesis tasks. Typically, such a system will accept some specification of the design problem as the initial working facts, and proceed to 'infer' a design solution from these. Most of these systems address configuration design problems: this is probably due to the fact that the available declarative knowledge of the domain components facilitates the expression of the heuristics, in the manner described in the previous chapter.

This section describes some of these systems (including several addressing the task of designing fluid power systems), and identifies some of the main design problem-solving ideas that have arisen. In every case, the heuristics used by these systems have been acquired through the use of conventional knowledge engineering. This is not intended to be a comprehensive survey, but rather to provide a flavour of approaches adopted and the tasks to which they have been applied, and also to indicate some of their recurrent shortcomings.

Functional Reasoning

A theme common to a number of these design KBSs is *functional reasoning*. Put simply, this approach tackles the design problem by explicitly considering and reasoning about the functionality that a solution must provide. The functionality implied by the specification must be matched to solution elements which embody this; as a consequence, the system must be given (to be able to infer) descriptions of both the specifications and potential solution elements at a functional level. This is not an easy task, since this functional level is a human construct, resulting from the manner in which artefacts are used. Furthermore, the appropriate degree of abstraction of this functional level for a given task is rarely obvious.

One system employing functional reasoning, for the configuration of microprocessor systems, is MAPLE (Bowen, 1985). During the design process, a component is selected and combined with another on the basis that it provides some functionality required by the second component. When the functionality of the combination of components matches that required by the specification, the configuration is thought complete. This approach demands that the functionality of components is well known, as is the cumulative effect of the design choices in terms of the functionality of the whole. (Compare the rule quoted above from the R1 system; in contrast to MAPLE, R1 configures computer systems without recourse to explicit expressions of functionality.)

The MICON tool (Birmingham *et al.*, 1988) reasons about the function and design of sub-systems of computer hardware design, and combines these to form an integrated system. It uses the concept of *templates*, or portions of a design common across solutions, in building this hardware system. These templates represent the manner in which sub-solution elements are connected. Chapter 5 will discuss the use made of configuration templates in representing fluid power systems for the purposes of this work.

A functional reasoning-based approach to the configuration of fluid power systems is found in the work of Kota and Lee (1993a; 1993b), whose technique involves defining a solution at a highly abstract (possibly generic) level and gradually refining the decisions made until the solution is instantiated with specific components. This requires highly detailed and structured domain knowledge in the form of abstraction hierarchies of components. Functional reasoning approaches to the configuration of fluid power systems can also be found in the rule-based systems developed by Lin and Shen (1995), Westman *et al.* (1987) and da Silva and Dawson (1997).

Welch and Dixon (1994), amongst others, maintain that, rather than translating directly from function to form, the design reasoning can usefully take place at the intermediate level of physical behaviour. Ulrich and Seering (1989), for instance, translate low-level descriptions of the relationship between input and output quantities into physical solutions that can achieve this. Their design technique is to generate a candidate design solution, derive the behaviour of this solution, and, based on this behaviour and knowledge of the domain, try to modify the solution so that it more exactly meets the specification. However, this takes place at the level of relatively simple physical systems; behavioural reasoning for design might not be tractable, nor, indeed, useful, when considering more complex systems.

Constraint Satisfaction

Certain design problems can usefully be viewed as constraint satisfaction (CS) problems. In general terms, the goal of CS is, simply, to find some solution that satisfies a given set of constraints (Rich and Knight, 1991). A CS problem is defined by:

- a set of variables;
- for each variable, a function that maps it onto a domain (i.e., which determines its value), and;
- a set of constraints.

Each constraint serves to restrict in some way the values that the set of variables may assume. A solution to a CS problem consists of the assignment of a value from its domain to each variable in such a way as meet the given specification, while violating none of the constraints. The specification may itself be stated in the form of constraints upon a solution, or perhaps as the assignment of fixed values to some subset of the variables. A consistent set of value assignments might not always be possible, in which case some relaxation of specification or constraints may be made.

The process of CS itself typically involves making some initial assignment of a value to a variable, and propagating the effect of this throughout the system: this will have the effect of constraining the values that other variables can take. If none of the constraints has been violated by this choice, a further choice is made, and is propagated, and so on, until either a constraint is violated or else, consistent values for every variable have been found - these values constitute the solution to the problem. In the former case, one or more of the value assignments will need to be undone, and alternatives tried. Hence, CS can be seen as a search for a set of values consistent with the set of constraints placed upon them.

The heuristics in a CS KBS reside in the constraints themselves and in the criteria for making and undoing assignments to variables. The 'network' of constraints can represent quite refined and sophisticated heuristic knowledge of the domain, knowledge which influences and guides the search towards a viable solution. A major limitation of the CS approach, however, is that this sort of problem-solving would seem to lend itself most naturally to problems describable in purely quantitative terms.

The VT system (Marcus *et al.*, 1988) for configuring elevator systems has become something of a benchmark in studies of configuration design automation, having been tackled using several different approaches. In the initial implementation, knowledge of elevator systems is stored as a network of interrelated constraints. For example, one constraint expresses the knowledge that:

> *"there must be at least an 8-inch clearance between the side of the platform and a hoistway wall and at least 7 inches between the platform side and a rail separating two cars."*

As above, when such a system has been designed that satisfies the design specification and violates no constraint, then the design process is complete. If a violation is encountered, the system is able to backtrack, using knowledge of the domain, to a position at which a design decision caused the infringement and make an alternative choice. This model ensures that

the task is essentially one of selecting a coherent set of numerical parameters to satisfy the current performance requirements.

Interestingly, VT also incorporates a knowledge acquisition tool whereby, in the event of the failure to produce a configuration, the system recognises that a weakness exists in its knowledge and prompts the user for more information. This is then stored in the appropriate manner at the appropriate location in the knowledge base for future use. This is tantamount to an admission of the incompleteness of the knowledge, and is an imaginative approach to addressing it during system development, when an expert takes the place of the user, and corrects the knowledge base in this manner.

The COSSACK system (Freyman and Mittal, 1987) employs a similar constraint-satisfaction approach to the configuration of microcomputer systems, while the ACDS system (Darr and Birmingham, 1994) provides a generic approach to CS problems. The PRIDE system (Mittal *et al.*, 1986) decomposes the design problem into a number of lesser tasks, each of which can be solved by an associated CS method. Medland and Mullineux (1993) discuss the design of mechanisms through constraint satisfaction.

However, a shortcoming of this sort of approach for conceptual design is that, to a certain extent, decisions need to have been already made about the content of a solution. It is necessary to know what parameters constitute a solution before values can be assigned to them and the constraints that exist between them can be defined. So, while the task can be applied to systems in which, say, the configuration of solution components is fixed, and the task is one of selecting consistent parameters for those components (as is the case in the above systems used as examples), it is more difficult to apply to the selection of the components in the first place. Some work has been done concerning dynamic CS systems, which amend the set of constraints.

A further difficulty is that of the problem specification. Design specifications typically include qualitative values, such as descriptions of desired functionality. It is not easy to see how these can be incorporated into CS problem descriptions. Both these difficulties would seem to suggest that the use of CS in design is limited to detail and embodiment design tasks, or else to optimisation problems, in which the major conceptual decisions have already been made.

Agent-Based Design

AIR-CYL (Brown and Chandrasekaran, 1986), a system for designing air cylinders, tackles a design problem through the use of a hierarchical community of design agents, each with a

repertoire of design methods to accomplish a particular task. At the root of this hierarchy is an abstract agent that governs the entire design process, with lower agents becoming gradually more specialised in a particular area of the design. Within this framework, the principal technique used in this case is again one of constraint satisfaction. The inability of a particular agent to overcome a local difficulty results in control being returned to its immediate parent agent, and a different agent being invoked.

Agent-based design represents a high-level strategy, exploiting heuristic knowledge about manner in which the problem can be decomposed into a set of separate, lower-level tasks, which may be solved through the use of quite different approaches. Whereas the other systems discussed in this section address their design tasks with the application of a single approach, this use of agents recognises that a single uniform approach, at one particular level of abstraction, might not be the best way of tackling some design tasks.

3.6 Case-Based Reasoning

Aside from KBSs, the second principal approach to the implementation of heuristic reasoning is Case-Based Reasoning (CBR) (Riesbeck and Schank, 1989). As seen above, a KBS uses explicit problem-solving heuristics, stored in a knowledge base. In contrast, a CBR system makes use of a store of explicit previous problem-solving episodes – or *cases* – to address a new problem. Each case consists of a description of a problem and its corresponding solution; together, the cases constitute the *case base* of the system. When posed a new problem, the CBR approach involves searching the case base to find that stored problem which has most in common with the new one. This judgement is based upon some *similarity metric* known to the system. The solution to this matched problem is proposed as the basis of a solution to the new problem. In this way, a solution is proposed by analogy to the manner in which a similar problem was solved in the past.

The next stage is to evaluate this retrieved solution. If it is not a satisfactory solution to the new problem, then it must be adapted in some way. Usually, this will be done using explicit adaptation heuristics, which modify the solution in some manner to make it more appropriate. In CBR, then, the heuristic knowledge in the system lies in the combination of the cases, similarity metric and adaptation knowledge. Figure 6 depicts the CBR approach.

The advantages of the CBR approach for design problems may be summarised as follows (Hua and Faltings, 1993):

- CBR does not require a complex domain model (as might be required for, say, a KBS addressing the same problem). Complete and complex design solutions can be produced even with a small knowledge base (but see the first difficulty below).
- The initial proposed solution is already a complete design, thus reducing the complexity of subsequent reasoning.
- As design cases are the source of knowledge, the expansion of the system is easily implemented as the storage of new cases. Also, the problems of the acquiring the knowledge would seem to be eased since much of it is expressed in terms of explicit problems and solutions, rather than the compiled heuristic knowledge of experts.

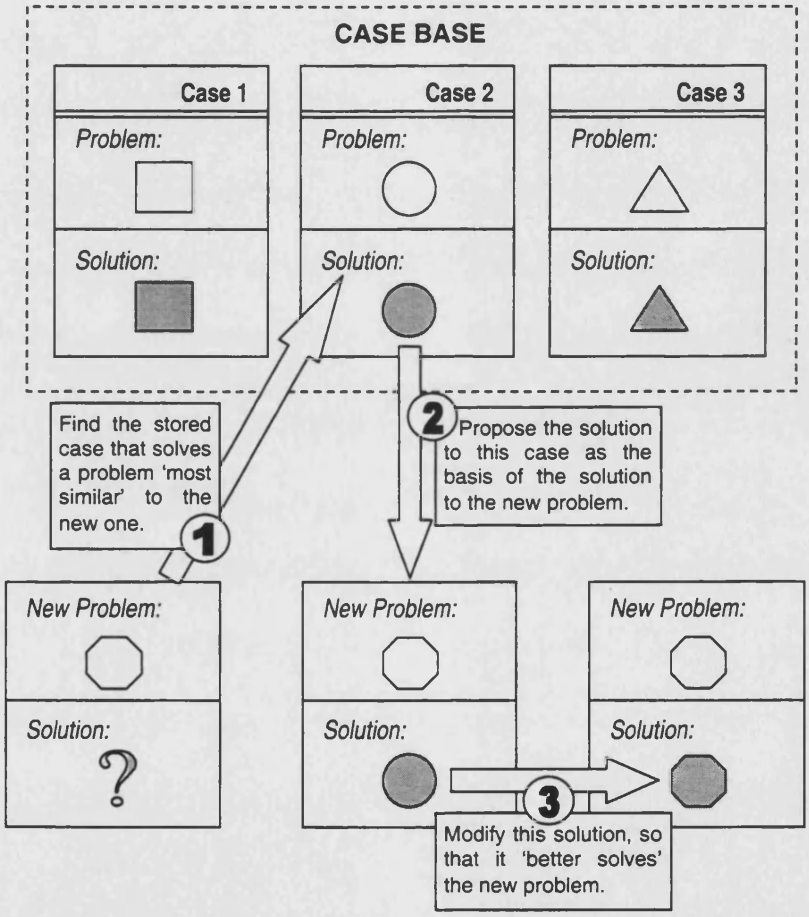


Figure 6. The Case-Based Reasoning approach to problem-solving.

However, the implementation of CBR systems is not without its complications: in particular, the following questions must be answered (Rich and Knight, 1991):

- *How are cases organised in memory?* The organisation of the case base becomes an issue with a large number of cases to search.

- *How are relevant cases retrieved from memory?* The similarity metric can be difficult to acquire and formulate. The features of the problem description relevant to determining similarity must be identified (and irrelevant features disregarded), and weighted in such a manner as to arrive at some comparable (and hence, numerical) figure expressing the degree of similarity. This quantitative measure does not seem to be particularly 'natural' knowledge, and a workable metric can often only be achieved through a process of trial and error, with an expert gradually altering the weightings until satisfactory performance is achieved.

- *How can previous cases be adapted to new problems?* Any solution based on an inexact match is unlikely to provide a wholly correct solution to a new problem, in which case, to make it acceptable it needs to be altered. This involves identifying the failures of the retrieved solution and then rectifying these. But, obviously, the system needs to 'know' how to do this, and failure diagnosis and repair is not a straightforward task. The system would seem to require a domain model and heuristics of a relatively complex nature to achieve this, undermining one of the perceived advantages of this form of problem-solving. The fewer the number of cases available, relative to the number of potential problems, the smaller the chance of finding a good match; and so, the greater the burden on this adaptation knowledge – it has to do more of the problem-solving work. Often the adaptation is performed by a KBS-type mechanism, and is thus vulnerable to the problems associated with these systems.

- *How are cases originally acquired?* In most domains, it would seem optimistic in the extreme to expect a ready-made, sufficiently large corpus of cases, each described in an appropriate manner, to be available. If CBR is to be applied, though, such a case base must be developed. Although the problems and their solutions should, by their natures, be more readily available than heuristic knowledge, these cases must be described in a suitable, consistent manner. This representation must be amenable both to judging the similarity of problems and to adapting the solutions.

Despite these difficulties, cases *do* seem to be a valuable source of problem-solving knowledge, and CBR has been applied to a number of different tasks.

3.6.1 Case-Based Reasoning Systems in Engineering Design

The CBR paradigm, then, is based upon the use of previous, successful problem-solving episodes to solve a new problem. This is analogous to the re-use of previous design experiences to solve new design problems, a strategy that seems to be quite natural, corresponding to adaptive or variant design. This has resulted in great deal of research into

CBR design tools. Again, this section presents an overview; Maher and Garza (1997) and Maher, Balachandran and Zhang (1995) provide greater detail on the topic.

At the highest level, many approaches conform to a strict two-stage model of CBR: a design case is retrieved from the memory of cases, and then evaluated and adapted to the needs of the current problem. A case is usually indexed and then retrieved on the basis of its design specification, although, in some instances, a lower level, behavioural description of the artefact is used. The adaptation is usually performed using some generalised, heuristic knowledge of the domain in question. Maher *et al.* (1995) make the point that:

> *"Case adaptation can be simply defined as making changes to a recalled case so that it can be used in the current situation. Recognizing what needs to change and how these changes are made are the major considerations. Adapting design cases is more than the surface considerations of making changes to the previous design, it is a design process in itself."* (p. 109)

This adaptation can be implemented in a number of ways. For example, Kritik2 (Goel *et al.*, 1992), developed for designing physical devices, uses detailed *structure-behaviour-function* models of its cases to identify discrepancies between the retrieved case and the current problem, and then *repair plans* to modify the case accordingly. PANDA (Roderman and Tsatsoulis, 1993), an automated assistant for designing fire engines, can adjust the values of parameters, or add or replace parts of a solution. CADRE (Hua and Faltings, 1993) attaches adaptation knowledge to each case, in terms of constraints on dimensions and topological change rules. The DEJA VU system (Bardasz and Zeid, 1993) uses a blackboard architecture, integrating disparate elements of unrelated cases, to plan and implement case modifications.

As an alternative approach, rather than being used to adapt solutions, generalised knowledge is used in the case-retrieval mechanism to attempt to identify useful (and compatible) elements of a number of cases, which can then be combined to construct the new solution. This technique has been termed *constructive CBR* (Pu, 1993). The PANDA tool uses this tactic when a suitable individual case cannot be found. Certain concepts are shared by the approach of Navinchandra *et al.* (1991), whereby a description of the behaviour of a desired hydro-mechanical system is transformed into an equivalent description that matches some combination of cases in memory. COMPOSER (Purvis and Pu, 1998), applied to configuration design, combines information from a number of cases: this information is integrated using a CS methodology.

Throughout the above approaches, however, there is a marked tendency to utilise quite detailed and low-level domain knowledge in order to adapt or combine cases. Modelling a domain completely enough to permit this is difficult, and negates one of supposed advantages of CBR, namely, the use of 'shallow' domain knowledge to produce realistic solutions. Later in this thesis, in chapter 11, a constructive CBR approach to fluid power systems design will be presented that employs high-level, generalised knowledge, of a relatively simple form. Although this occurs within a simplified and restricted domain, the system operates using an uncomplicated domain model, which facilitates its successful acquisition and expression, and which should ease subsequent maintenance.

3.7 Non-Heuristic Approaches to Design Automation

In contrast to the heuristic approaches to design automation seen above, there have been several approaches to design in which no attempt is made to express or use heuristic knowledge. Instead, these approaches tackle the design problem by making use of the particular strengths of digital computers, such as fast processing times and large memories.

In theory, the design process could be automated, with guaranteed success, if the design domain had the following features:

- a tractably finite number of potential design solutions;
- an analytical algorithm by which the performance of each solution could be determined, and;
- design specifications couched at the level of this deducible performance.

Then, given a new design specification, it would be possible to satisfy it by generating each solution in turn, analysing it and comparing the predicted performance with the desired. A match would indicate that a design solution had been found. In complex domains, having many potential solutions, this would be a purely computational approach: no human would contemplate the sort of search that this strategy entails. However, its usefulness as a computer-based model of design is also limited. Many domains do not have a tractably finite number of potential solutions, and, in general, design specifications are stated at a higher, more human, level (i.e., at the level of functionality) than the description of behaviour that is usually provided by analytical simulation tools.

However, this sort of approach has been applied to limited problems. For example, a *genetic algorithms* (GA) method (Goldberg, 1989) has been used to solve design problems. The technique is one of 'evolving' solutions, by analogy with natural selection processes. A

population of potential solutions is generated, usually at random, and each is evaluated using a 'fitness' function. The fitter members of the population are permitted to 'reproduce': this is done by combining features of these members, thereby creating new potential solutions, and a new population. With an element of randomness added (a small percentage of solution features are altered at each step), this process continues until the population converges, that is, the majority of solutions are identical - and, hopefully, good. The evolutionary pressure forcing solutions towards this single, good solution is provided by the fitness function. By encoding potential design solutions in an appropriate manner, and incorporating the current design specifications into a fitness function, this technique has been applied to design problems by, amongst others, Brown and Hwang (1993).

However, as with constraint satisfaction, this approach would seem to be limited to optimising solutions in which the major conceptual decisions had already been made, since the reproductive processes generally require solutions to be described using a consistent number of features. In addition, a GA approach requires a quantitative fitness function expressing the criteria for a good design.

As the GA approach relies on generating a large population of initial candidate solutions and typically requires a large number of iterations before convergence, once again this is not an approach that a human expert would envisage adopting. Schmidt and Cagan (1995) describe a similar design method which uses *simulated annealing*, another optimisation technique.

Although it is included here for the sake of completeness, the usefulness of a non-heuristic approach would seem to be limited. For the design of fluid power systems, the number of solutions is theoretically infinite (another component can always be added to a configuration), and, although analytical simulation tools do exist, they produce low-level analyses of behaviour, and design specifications are generally expressed in higher terms of functionality. No simple, numerical solution-quality metric is available.

No further consideration will be given to non-heuristic strategies in this thesis. This research is focused upon capturing the sort of heuristics that are employed by KBS and CBR systems: this would seem to be a more profitable avenue of research given the nature of the working domain.

3.8 Summary

The key to performing complex and difficult tasks successfully lies in the ability to access and apply the appropriate knowledge at the appropriate times. Accordingly, the content and

representation of knowledge in intelligent systems has been a principal theme of AI research.

In this chapter, some of these ideas have been introduced, as has a model of the types of knowledge in a design synthesis system. Design synthesis relies to a great extent on knowledge of a *heuristic* nature, developed as a result of the designer's experiences. For building intelligent computer systems, the conventional approach to acquiring the required knowledge has been through a process of *knowledge engineering*. However, there are a number of serious limitations to this approach, which call into question the accuracy of the derived heuristics – and of the Knowledge-Based Systems that incorporates and reasons with this knowledge.

A second general approach to design automation is through the application of Case-Based Reasoning. To a certain extent, this overcomes the problems associated with acquiring explicit heuristic knowledge, since this knowledge is stored in the form of explicit examples of previous successful problem-solving episodes. These examples are used to propose solutions to new problems. However, explicit heuristics are required to access these examples, and then modify them to meet the new problem.

Since this heuristic knowledge would seem to be essential for successful design synthesis, the difficulties presented by a knowledge engineering approach form a serious obstacle to the development of automatic design systems. The following chapter discusses one possible solution to the difficulties of knowledge engineering – *inductive machine learning*.

4 Inductive Machine Learning and Design

Learning is often viewed as a fundamental aspect of intelligence, since it enables the learner to gain autonomy in a task environment, ensuring that its success is not wholly dependent on the knowledge provided by its creator (Russell, 1996). This becomes especially important when the knowledge of the creator is imperfect or incomplete.

Learning can be thought of as:

> *"...changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time."* (Simon, 1983).

It may also involve the process of updating and revising the manner in which knowledge is stored to achieve the same ends (Stillings *et al.*, 1987). Defined thus, the general term 'learning' can be seen to cover a wide range of tasks, from learning to ride a bicycle or how to play chess, to memorising useful telephone numbers.

As would be expected, research into emulating learning processes on computer forms a large part of AI research – this is the study of *machine learning* (ML). The idea of a computer system that is able to learn is very appealing. However, like AI in general, the field of ML is still relatively immature (the first ideas appearing in the 1940s, with the majority of work being published in the years since the mid-1970s) and a general learning mechanism is not imminent. That said, the techniques already developed have had some success, and their results have been of interest to psychologists and cognitive scientists, since the understanding of human learning is still limited. More recently, these techniques have been used in *data mining* applications to search for useful regularities in the historical data of organisations (Mitchell, 1999).

Buchanan and Shortliffe (1984) identify three ways by which knowledge may be acquired for intelligent computer systems:

- *handcrafting* – the expert is also a programmer, and is able to code his/her knowledge directly.

- *knowledge engineering* – work with an expert to acquire and organise the required knowledge.
- *machine learning*.

In general, handcrafting can be discounted, since finding an expert with the right combination of skills is unlikely for most tasks. The knowledge engineering approach, as seen earlier, is susceptible to the knowledge bottleneck problem, especially when trying to acquire heuristic expertise. The final alternative, ML, would seem to offer a practical approach by which this bottleneck might be circumvented: by automatically acquiring the heuristic knowledge required for intelligent systems, there would be less dependence on interviews with human experts for the knowledge. As Reich and Fenves (1989) remark:

> *"The incorporation of learning techniques into [intelligent] systems has the potential of alleviating the knowledge acquisition bottleneck and supporting performance improvements over time, thereby providing a more promising future for [intelligent] systems technology."*

This realisation has stimulated research into ML in the past (both Michie (1982) and Lenat (1983), for instance, highlight the importance of this aspect of the research), and it stimulates the research described in this thesis.

In general, a particular algorithm or approach may be characterised as performing either *inductive* or *deductive* learning. As the name suggests, inductive learning involves drawing, from a limited set of examples, general conclusions that are asserted to hold true for all examples of the same type. In contrast, deductive learning involves analysing known facts and relationships to derive more knowledge. The majority of research to date has been directed at inductive learning (Bratko, 1993), and since this research concerns the inductive acquisition of design knowledge from design examples, the following discussion will be restricted to inductive algorithms. (For a more general overview of approaches to ML, see (Carbonell *et al.*, 1983) and (Winston, 1993).)

4.1 Inductive Machine Learning Algorithms

Inductive algorithms accept data consisting of a number of examples of some concept (this data is termed the *training data*). If the learning is to be *supervised*, an example comprises an input pattern and a corresponding output pattern: the learning task will be to infer the relationship that holds between the two. If the learning is *unsupervised*, an example consists of an input pattern alone: the task is to search for the implicit relationships that exist within the data.

In general, each example is described by the values that it possesses for each of a number of attributes. Since the task of learning knowledge may be viewed as one of recognising consistent relationships amongst the attributes, this is only possible if these attributes are themselves described consistently. Consequently, each example must be described using these attributes, and their values, in a consistent manner. *Symbolic* ML algorithms operate using qualitative values, whereas *subsymbolic* algorithms expect quantitative values (with some algorithms able to accept a mixture of value types).

Successful inductive learning algorithms must be able to *generalise* appropriately over their training data. It would not be acceptable to, for example, simply produce a look-up table of the example inputs and their corresponding outputs: this involves no learning beyond memorisation by rote, and would not be able to respond successfully to examples other than those in the training data. However, the algorithm alone does not determine the quality of a learned generalisation: the training data must be representative of the domain, and sufficient in both quantity and quality to permit this.

Many different algorithms for inductive ML have been developed; in general, each can be considered to fall into one of several groups: *classification construction*, *conceptual clustering*, *associative learning*, *inductive logic programming* or *artificial neural networks*. The following sections discuss each of these groups in turn.

4.2 Classification Construction

This involves the learning of classification knowledge; in other words, knowledge that will allow an unclassified new example to be appropriately categorised into one of a number of classes. This is a supervised learning task: this knowledge is learned from a set of training examples, each labelled as belonging to a particular class. Hence, each example consists of a set of value-attribute pairs, plus a class label.

To illustrate this idea, a series of examples might describe the weather conditions on certain days (this example is adapted from (Quinlan, 1986)). Based on the values of certain attributes describing the weather, each example is classified as being either appropriate or not for some unspecified purpose. The learning task is to acquire the criteria necessary for making this classification.

A sample of the training data might look something like that shown in Table 2. Four attributes, plus a class attribute, are used to describe each example. Each attribute can assume one of a number of values – for instance, *outlook* can be *overcast*, *sunny* or *rain*. The

class attribute indicates which of the two possible classes – *P*, positive or *N*, negative - the example falls into.

	attribute				
example	*outlook*	*temperature*	*humidity*	*windy*	*class*
1	overcast	hot	high	false	P
2	sunny	hot	high	false	N
3	sunny	hot	high	true	N
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	sunny	mild	normal	true	P
...					

Table 2. Example training data.

The learned classification criteria might take the form:

if outlook = sunny *and* humidity = normal
or if outlook = overcast
or if outlook = rain *and* windy = false
then class = P
else class = N

This is the knowledge, here in the form of a rule, which allows a new day to be classified according to its weather. It consists of a series of tests on the values of attributes; based on the values that a new example possesses, it will be placed in one class or the other. This induced knowledge need not be represented as a rule; it could, for example, take the form of a decision tree.

Typically, this sort of induction will be generated in either a top-down or a bottom-up manner (Gillies, 1996). The former approach involves selecting some very general induction, and gradually specialising it to cover more examples. Such an approach might begin, on the basis of the first example, by proposing the rule:

if outlook = overcast
then class = P
else class = N

This rule is then tested on the remaining examples. Whenever it misclassifies a negative example, this means that the conditional statement testing the attribute values is too general – it includes an example that should be excluded – and so it needs to be specialised to omit this case. Conversely, whenever a positive example is misclassified, this means that the conditional statement is too specific, since it excludes this positive example. Hence, it needs to be generalised to include this case.

A bottom-up approach, on the other hand, would be to start with a specialised induction, and gradually generalise it. Based on the first example, a specialised rule might be:

if outlook = overcast *and* temperature = hot *and* humidity = high *and* windy = false
then class = P
else class = N

This rule then tested on the remaining examples, and developed as for the top-down approach. It is the criteria for selecting and modifying the rule premises that determine the nature and quality of the knowledge that will eventually be generated.

4.2.1 The *CN2* Algorithm

One particular classification construction algorithm is *CN2* (Clark and Niblett, 1989; Clark and Boswell, 1991). As will be discussed in chapter 8, this algorithm has been applied to the task of learning design synthesis knowledge, and so is described in some detail here.

This is a symbolic rule induction algorithm, attempting to find classification knowledge in the form of rules, similar in form to those given above. So, each of these rules has the structure:

if <condition>
then <classification>

Again as above, a rule condition takes the form of a test for the presence of particular attribute values in an example. So, if a new example has a matching combination of attribute values, then it may be classified as indicated.

Given a number of pre-classified training examples, the CN2 algorithm proceeds in an iterative fashion to find a good set of rules for classifying examples. This is performed in a top-down manner by starting with a set of rules having conditions of a very general nature, testing these, and gradually specialising the better rules in an attempt to find still better ones.

A rule is tested according to two measures. First, an error measure is applied to evaluate the *goodness* of the rule. This measure favours rules that correctly predict a large number of a single class and few of other classes. Secondly, the *significance* of the rule is determined - a rule is considered significant if it describes a regularity in the data that is unlikely to have occurred by chance. This is done by comparing the class frequency distribution produced through the application of the rule with the distribution that would be expected had the classification been made at random according to the frequency of classes in the example

data. The greater the difference between the two distributions, the more likely it is that the rule reflects a significant regularity in the data. Rules that are found to have both high *goodness* and high *significance* are more readily adopted into the set of learned rules than are those having lower values of one or both of these measures.

4.3 Conceptual Clustering

Conceptual clustering algorithms attempt to produce some sort of description of a domain from a set of data exemplifying it; typically, this description takes the form of a taxonomy or hierarchy (Gennari *et al.*, 1989). Items towards the top of this hierarchy are more general domain concepts - the single node at the top of this hierarchy (the *root* node) will be the most general concept, describing all of the data. Nodes below the root get successively more specific, referring to more specialised concepts within the data. Nodes at the same level in the hierarchy describe alternative concepts at that level. Terminal nodes describe the most specific concepts - usually, these will be the examples themselves. These algorithms work by searching for similarities within the data. Examples that share some similarity are clustered together, and a description of this similarity is used to describe the concept so formed.

4.3.1 The *COBWEB* Algorithm

One example of a clustering algorithm is *COBWEB* (Fisher, 1987). COBWEB attempts to construct a conceptual hierarchy from a set of examples, each of which is described in terms of a set of attribute values. Each concept, or 'category', in the hierarchy is expressed in terms of probabilities; an example member of that concept will have a certain probability of possessing each of the possible attribute values. These values allow a new example to be categorised into that concept at each level of the hierarchy to which, on the basis of its attribute values, it 'most probably' belongs.

The algorithm functions in an incremental manner. The first example is used to define a root node. Each subsequent example is added to the root, modifying the probabilistic description accordingly. It is then recursively classified into one of the child categories (thereby modifying its description), or, if the example is not sufficiently similar to an existing category, by creating a new one to house this example. Eventually, a category describing this example alone will form a terminal node to the hierarchy. Additional operators can merge two categories into a single category, or split a category into several. These help to revise the hierarchy in the light of new examples that have the effect of invalidating the classifications of previous examples.

The similarity is determined using a *category utility* measure. This measure, derived from information theory, has been found to have some psychological basis in category formation. The measure is based upon the probabilities of attribute values; examples having similar probabilities of particular values of attributes are more likely to be describing the same concept.

COBWEB is an unsupervised learning algorithm; the hierarchy construction is based on the similarities amongst the attribute values of the data – no prior classification or output associated with the data is used during learning.

4.4 Associative Learning

Associative learning is another form of unsupervised learning. The task is one of recognising mutual occurrence patterns in a body of example data. The archetypal problem of this sort is 'shopping basket analysis': given a number of examples of the contents of shoppers' baskets, can any patterns be recognised in the types of goods that are bought together? Analysis might reveal the associative rule:

bread, cheese, milk $\Rightarrow$ eggs

This can be read as follows: "there is evidence in the data to suggest that in cases in which bread, cheese and milk appear in the shopping basket, it is likely that eggs also appear". The purpose of associative learning algorithms is to find rules of this sort, based on certain ideas of what constitutes sufficient evidence for an association.

4.4.1 The *Apriori* Algorithm

One such associative learning algorithm is *Apriori* (Agawal and Srikant, 1994). In more formal terms, the problem may be stated as follows. Let $I = \{i_1, i_2,...,i_m\}$ be a set of symbols, representing all the items that can possibly occur in an example. Let D be a set of examples, where each example, E, is a set of items such that $E \subseteq I$. For X, some set of items in I, if $X \subseteq E$ it can be said that E *contains* X. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$ and $X \cap Y = \varnothing$. The rule $X \Rightarrow Y$ holds in the set D with *confidence* c if $c\%$ of the examples in D that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s in the set D if $s\%$ of examples in D contain $X \cup Y$.

The learning task is to generate all the association rules that have values of support and confidence greater than prescribed minimum values. Note that it is important that rules exceed both of these thresholds, since the appearance of some particular combination of

items in only one example in the data would in all probability be merely coincidental, yet could produce associations of 100% confidence. A greater degree of support across the data is required.

With large numbers of examples, containing large numbers of items, the potential number of association rules is also large. The Apriori algorithm uses the measure of support to control the search for valid rules. Simplified, the algorithm proceeds as follows:

- First, all the sets of items that have support equal to or greater than the minimum are found.
- Secondly, these sets are used to generate the desired rules. For each set, L, find all the non-empty subsets of it. For each such subset, A, generate a rule of the form $A \Rightarrow L \setminus A$ if the ratio of the support of L to the support of A is greater than or equal to the minimum required confidence.

4.5 Inductive Logic Programming

The approaches already described in this chapter employ *attribute-based* learning; in other words, the example data and resulting knowledge is described in terms of attribute-value pairs (for example, *temperature = hot*). While this is adequate when learning about independent attributes, it is not possible to express relationships between attributes (for example, to express the fact that attribute *part1* is next to attribute *part2*) in this manner. In addition, it is difficult to express background knowledge (that is, existing knowledge that may help to guide the learning process) in these approaches. Potential roles for the background knowledge might include the expression of structural or topological relationships, existing models, or known facts and laws (Bratko, 1993).

These problems have led to a number of algorithms which learn at the level of first-order predicate logic – collectively, this area of machine learning is called *Inductive Logic Programming* (ILP) (Muggleton, 1991). A learning problem in ILP is characterised in the following manner (Bratko and Muggleton, 1995): given background knowledge B, expressed as a set of predicate definitions, positive examples, E^+, and negative examples, E^-, an ILP system will attempt to construct a logic formula H, such that:

- all the examples in E^+ can be logically derived from $B \wedge H$, and
- no example in E^- can be logically derived from $B \wedge H$.

Typically, B, H, E^+ and E^- will each be logic programs (and the problem can be generalised to encompass more than two possible classes of outcome). So, background knowledge can

be introduced into the learning task, and, since the logic programs are described in first-order predicate logic, relationships between attributes can be stated as n-place predicates. For instance, introducing a 2-place predicate *next_to(x,y)*, it is now possible to describe the relationship *next_to(part1, part2)*; attribute-based learning is, in effect, limited to single place predicates, such as *temperature(hot)*.

4.5.1 The *GOLEM* Algorithm

GOLEM (Muggleton and Feng, 1992) is one such ILP algorithm. It works by constructing a generalisation of pairs of examples in turn. To try to avoid making large inductive leaps, which may not be justified by the other examples, the most specific 'useful' generalisation is formed; hence its name, a *relative least general generalisation* (rlgg). Each of the examples is described in terms of a conjunction of logical clauses; as a result, the rlgg is described in similar terms. The rlgg that correctly classifies the greatest number of examples and misclassifies the least is chosen, and then, in combination with each of the remaining examples in turn, is used to construct a further rlgg, and so on. In this bottom-up manner, a rlgg that 'explains' all or a sufficient number of the examples is formed – this rlgg is a *logic program*, which, it is suggested, has generated these examples.

As an example of its relational learning, this technique has been used to learn the rules for creating appropriate finite-element meshes for cylinders from hydraulic presses (Dolsak and Muggleton, 1992), a task usually requiring a degree of expertise on the part of an engineer. One of the logic program 'rules' that was induced, given here in the PROLOG logic notation that the algorithm uses, is:

```
mesh(Edge, 7) :-
    usual_length(Edge),
    neighbour_xy(Edge, EdgeY),
    two_side_fixed(EdgeY),
    neighbour_zx(EdgeZ, Edge),
    not_loaded(EdgeZ).
```

In English, this states that a certain edge should be partitioned into 7 mesh elements if it is of usual length, and it has a neighbouring edge in the y-plane that is fixed on both sides, and it has a neighbouring edge in the z-plane that is not loaded. This sort of rule cannot be expressed (and therefore, cannot be learned) in attribute-learning systems.

4.6 Artificial Neural Networks

Artificial Neural Networks (ANNs) are inspired by models of the brain, and in particular the manner in which large numbers of signal-passing neurons, each displaying relatively simple behaviour, can together produce complex behaviour and responses.

An ANN consists of a network of interconnected mathematical units. Each connection has an associated numeric weighting which serves to 'amplify' or 'diminish' the strength of the numerical signals that are passed along it. The *activation* of a unit is a function of, typically, the sum of the weighted signals that it receives. This activation in turn influences the strength of the signal that the unit transmits along connections to other units. Following the activation of certain units by the application of some external stimulus, the signals in the network are allowed to achieve equilibrium. When this state is reached, the activation levels of some or all of the units form the response of the network to the stimulus.

There are a great number of different network topologies, each displaying different characteristics (for an overview, see (Lippmann, 1987)). A network can be 'trained' to represent some particular function or concept through the adjustment of its connection weights in response to example data. This network training can be either supervised or unsupervised, and can be applied to a number of different tasks, including learning classification knowledge, clustering data, and finding associations amongst data. However, because of the distinctive learning concept of adjusting weightings on connections between mathematical units, it is convenient to treat ANNs as a separate class of learning mechanism.

4.6.1 The *Backpropagation* Algorithm

One commonly used network topology is that of the *feed-forward* network. In this some units are designated as 'inputs', receiving some initial stimulus from the external environment, while others are designated 'outputs' indicating the final response of the network to its input when the sequence of unit calculations has settled into equilibrium. The input units together form the input 'layer' of units, and the output units the output layer. Between the inputs and the outputs may be any number of additional layers of units. Each may contain any number of units, and, in general, units in one layer tend to be connected to all those in the subsequent layer, moving from inputs to outputs.

A feed-forward network may be trained to respond to certain input patterns by producing some associated output. One supervised learning algorithm for doing this is the *backpropagation* algorithm (Rumelhart and McClelland, 1986). Using this, the network is

trained by repeatedly presenting examples of the correct combinations of inputs and outputs to the network, and gradually altering the connection weightings so that the input produces the desired network output for every example. Learning parameters govern, amongst other things, the rate at which weightings are modified, and the point at which training is considered complete. Some associations are more complex to learn and represent in this way; these associations tend to require networks with greater numbers of intermediate layers and units. It is found that, under the correct conditions, an ANN trained in this manner can learn a generalised representation of these associations. In other words, the network can respond appropriately to input data other than those used during training.

In this way, feed-forward ANNs have been found to successfully approximate functions, learn classification and pattern match. When trained, then, its knowledge is embodied in the form of the trained network itself. As may be expected from the above description, ANNs are subsymbolic learners: they expect their data to be represented numerically (and typically as normalised values between 0 (or –1) and 1).

4.7 General Considerations for Inductive Machine Learning

Beyond the characteristics of any particular learning algorithm, if anything of use is to be learned, there are a number of general considerations to bear in mind when applying inductive learning.

4.7.1 Data Representation

Typically, the examples presented to the algorithms must be described in terms of consistent sets of attribute-value pairs. In other words, all members of the same set of attributes are used to describe every example. The choice of attributes to describe a problem, and the values that each can take, must, as yet, be done by a human before learning commences. If useful inductions are to be made, it is crucial that the examples are described using relevant terms, and, as far as possible, omit irrelevant information that may lead to the formation of spurious associations.

During the discussion of induction in chapter 2, the need to control the inductions that can be made was stated: without this control, many associations of little use in the current situation could be inferred. For machine learning tasks, this control is effectively achieved by the manner of describing the examples, since the learned generalisation is expressed in terms of these attributes and values.

4.7.2 Data Quantity

If useful generalisations are to be learned, then there is a requirement that the example data be representative of the concept as a whole. However, it is virtually impossible to decide beforehand if a body of data is representative or not. In practice, this tends to be addressed by supplying as many data as are available to the algorithm. The amount required depends on both the complexity of the concept to be learned and the learning algorithm itself. Insufficient examples can result in little of use being learned.

The operation of a particular algorithm supplies an empirical response (albeit under highly controlled conditions) to the question raised in chapter 2 about the number of examples that are required to make a certain induction.

4.7.3 Data Quality

It is unrealistic to expect 'real world' examples to be correct and free from 'noise', so the performance of the learning algorithm when faced with this can be important. Certain algorithms are able to cope with noisy data better than others are, but, in general, the quality of the knowledge learned by all suffers.

Another important requirement is that the examples as a whole present a *self-consistent* description of the concept. An extreme case of inconsistency would be the appearance in the data of the same input pattern mapped to more than one output pattern, although subtler (and so, more difficult to detect) forms of inconsistency can occur. An algorithm might be able to cope with this as noise, but it might be indicative that the input attributes used are insufficient to predict the output, or even that the concept being modelled is non-deterministic, thus rendering futile any attempt to make inductive assertions about it.

4.7.4 Representation of the Learned Knowledge

The learned knowledge can be represented in different forms; in the algorithms described above, it is variously in the form of rules, concept hierarchies, logic programs and trained neural networks. When considering whether to use of a particular algorithm, thought should be given to the nature of the learning task, to whether the concept to be learned can be expressed in the representation formed by the algorithm, and to the intended use of the learnt knowledge.

4.7.5 Learning Bias

The operation of a particular algorithm will serve to focus the search for generalisations in a very particular way. This (necessarily) imposes a bias for learning certain things rather than others. Consideration of the sort of generalisations that might be useful can have a bearing on the type of algorithm that is suitable for a particular task.

4.8 The Application of Inductive Machine Learning

Although there are no strong methodologies for the application of inductive machine learning techniques, the process can be thought of as passing through the following stages (this is based upon a similar methodology suggested by Reich *et al.* (1993)):

1. The learning task must be thoroughly understood, and the choice made of an appropriate representation of the concept to be learned.
2. Examples of the concept must be collected and described according to the chosen representation.
3. A machine learning algorithm is chosen as being apt for learning the concept and, if necessary, the examples must be translated into a form that is compatible with the algorithm. With appropriate learning parameters, the algorithm is applied to the training examples.
4. The performance of the learned knowledge upon a subset of the data held back from training, or some other measure, is used to decide when the learning process has been successfully completed.
5. When a satisfactory body of knowledge has been learned, this can then be applied to the task of predicting the response to new examples of the concept.

It is often necessary to iterate around these stages until satisfactory performance is attained. The testing of the learned knowledge in stage 4 is vital if the extent and quality of its generalisation is to be examined. Typically, this is done by dividing the example data into two separate sets – training data and testing data. Only when the learned knowledge, formed using the training data, provides satisfactory performance on the testing data – by correctly classifying an acceptable number, for instance – can the algorithm be considered to have successfully learned generalised knowledge. If acceptable performance is not attained then it may be necessary to continue the learning process, if an improvement in performance is noticed, or else to repeat the learning attempt with different learning parameters or a re-division of the examples (the initial division may not have given a representative training or

test set). The failure of further attempts might indicate that the data are described using inappropriate attributes or values, or that the data are insufficient, or, in the worst case, that the concept is not learnable (at least not with this particular algorithm).

It is important to note that current ML techniques are *not* autonomous; a human is required to identify appropriate attributes and values and describe the data accordingly, and then to monitor and control the learning process. This identification of attributes and values is itself a form of knowledge engineering, as described in the previous chapter. However, Michie (1982) thinks that, in practical circumstances, this may not present a major problem, since:

> *"Even though he may not be able to tell you what to do with them, the expert...can usually supply a list of primitive features [i.e. attributes] which at least contains all those which are relevant, even though it may be padded out with additional features which the expert thinks are relevant but which are not. The expert typically possesses the further gift of being able to induce a grasp of the given concept in a trainee, by selecting and administering a well-contrived set of examples."*

The next chapter describes the attribute and value sets developed to depict the conceptual design of fluid power systems so that machine learning can be applied to learning about the task.

4.9 Machine Learning in Design

Engineering design synthesis knowledge is valuable, and difficult to express, so there would seem to be much to gain by attempting to use ML to capture this knowledge. As Russell (1996) states:

> *"Learning may...be the only route by which we can construct very complex intelligent systems. In many application domains, the best systems are constructed by a learning process rather than by traditional programming or knowledge engineering."* (p. 90)

Reich *et al.* (1993) identify the following, not necessarily independent, learning activities in and around the design process:

1. designers learn technical or analytical knowledge. For example, they learn how to make stress analyses of the structures that they have proposed. Typically, this is the sort of information taught in colleges and universities, and is that available in textbooks.

2. during the design process, "designers learn about the problem, its solution, and their relationship." In other words, the designer learns about the elements that describe the design task.

3. designers "assimilate experiences [of design episodes] for use in future design problems." Reich *et al.* continue:

   "These experiences are what differentiate expert and novice designers. Designers must always be aware that new design situations prevent the 'as-is' application of previous experiences. Designers need to learn the similarities as well as the differences between current and previous problems and adapt old solutions to new situations. There are currently no algorithmic solutions to this problem. What is needed is research that records and evaluates how current automated techniques can support such learning."

4. designers learn about the "viability of certain design beliefs, judgments, decisions, or practices in certain situations" through customer feedback and the success or failure of the constructed artefact in operation.

So, it can be seen that there is wide scope for learning in design contexts. This has led to a number of applications of (both inductive and deductive) machine learning to the task of learning design knowledge for intelligent systems. Duffy (1997) provides an overview of research into the application of machine learning techniques in design in general. Here, though, the interest lies in the acquisition of design synthesis heuristics (roughly equivalent to Reich *et al.*'s third and fourth learning activities above); consequently, the next section focuses upon previous approaches to automatically learning these heuristics.

4.9.1 The Inductive Acquisition of Design Heuristics

Given the difficulties surrounding the direct acquisition of design heuristics through knowledge engineering processes, machine learning techniques would seem to offer an attractive alternative: the automatic extraction of this knowledge from examples of design work. Several researchers have recognised this, and have experimented with inductive machine learning.

A theme common to a number of these approaches is that of conceptual clustering (described in section 4.3 above). Based upon recognised similarities within the descriptions of events (which, here, are design cases), these techniques construct a model of the domain represented by these events, usually in the form of a hierarchical structure of 'concepts', becoming gradually more specific to individual events as the structure is traversed. Reich

and Fenves (1992, 1995) use a clustering algorithm (*ECOBWEB*, an extended version of the *COBWEB* algorithm described in section 4.3.1) to create such a conceptual hierarchy representing the descriptions of bridge designs. Based on its design requirements, a new problem can be matched to the most similar cluster in the hierarchy, and the solutions to the examples stored under this cluster used as the basis for a solution to this new problem.

Other researchers to have used clustering approaches include Duffy and Duffy (1996) and Maher and Li (1992), who include a learning component that attempts to learn additional domain knowledge to assist future design. Lu and Chen (1987) have developed an 'intelligent decision-making framework' for their inductive inference method. They use simulations to generate raw examples of designs that are then clustered; the clustered data is then fed into a classification construction algorithm, similar to those described in section 4.2, to produce predictive rules for cluster membership.

However, there would seem to be a methodological problem with these clustering approaches. These algorithms search for similarities in the given examples in an unsupervised manner. When applied to clustering design specifications, there is no information about the design *process* to guide the learning, since the learned knowledge is merely a reflection of the similarities amongst the available design examples. This difficulty will be discussed in more detail in a chapter 9.

Aside from clustering approaches, Ivezic and Garrett (1994) use an artificial neural network (section 4.6) to try to predict the values of form and behaviour attributes of computer systems, whereas Coyne *et al.* (1993) use an ANN to try to complete partial designs of rooms, in terms of their contents.

Notwithstanding the promising results reported by these authors, there has been relatively little research into the automatic acquisition of synthesis knowledge. Since this knowledge is both valuable and scarce, this is a little surprising. The work has tended to be focused on small-scale, low-level tasks, and often using artificially created data. Based on this review, there would seem to be much fundamental research to be done in this area. Some of this research forms the basis of this thesis.

4.10 Summary

A number of different ML algorithms have been developed, addressing different learning tasks. As considered here, there are five general classes of inductive algorithms: classification construction, conceptual clustering, associative learning, inductive logic programming and artificial neural network algorithms. Each of these is able, after its own

fashion, to learn and represent generalised knowledge from a body of examples. As such, they would seem to offer a technique for overcoming the knowledge bottleneck in intelligent systems production. However, the 'learnability' of knowledge using this approach is dependent on a number of factors, including the choice of an appropriate learning algorithm, the amount and quality of the available examples and the representation chosen as the means of describing the examples.

There has been some previous research into the use of inductive ML techniques for the task of learning design synthesis heuristics. However, while some of this work has produced promising results, these attempts have often been limited in scope and fail to prove conclusively the worth of this approach.

Inductive machine learning has been applied in the course of the research reported here to the task of acquiring synthesis heuristics for the design of fluid power systems. Chapter 6 introduces the approach that has been adopted. Chapter 7 discusses some applications of feed-forward ANNs to the task of learning design heuristics, chapter 8 describes the application of a classification construction algorithm to this task and chapter 9 the application of a conceptual clustering algorithm. This is a real design problem, and the learning is to be based on examples of real design episodes. The purpose of these applications is to allow a general judgement to be made of the value of current ML algorithms for learning tasks of this sort.

First, however, a chapter is devoted to the fluid power systems design activity itself. In addition to describing this task in greater detail, the representations of the task that have been developed for use with ML algorithms will be presented, and the training examples that have been collected will be discussed.

5 Fluid Power Systems Design

This research is based around the conceptual design of fluid power systems. Fluid power systems convert rotational mechanical energy into fluid energy (via a pump), which is then transmitted by connecting pipes to some remote point(s) of actuation, where it is transformed back into mechanical energy. According to Henke (1983):

> *"The purpose of every fluid power circuit is to transfer energy to an output device, or actuator, for the purpose of doing useful work, such as moving a load"* (p. 13)

In addition to pumps and actuators, and the connecting pipe-work, there are a number of other standard domain components which serve to control the fluid pressure or flow rate, or to direct the flow, thereby producing some motion or force of a precise character at the actuator. A typical design specification, then, might describe the desired motion or force, and the design task is to configure some selection of the available components into the description of a system that will meet this specification. A simple fluid power system is shown in Figure 7. This figure uses a number of conventional symbols to represent the components and their connections. Characteristic applications of fluid power systems include earth-moving machinery, lifting apparatus and presses.

More generally, this design task is a configuration design problem: the task is one of choosing and connecting some set of pre-defined domain components to form a system that satisfies a given specification. A typical specification might describe functionality at the global level of the desired system, and can also impose operational, cost or other constraints on the task. Within these constraints, the designer can use any number of components, arranged in any manner, to solve this problem. Hence, according to the characterisation of such problems given in chapter 1, the design of fluid power systems is one of the more difficult configuration design tasks.

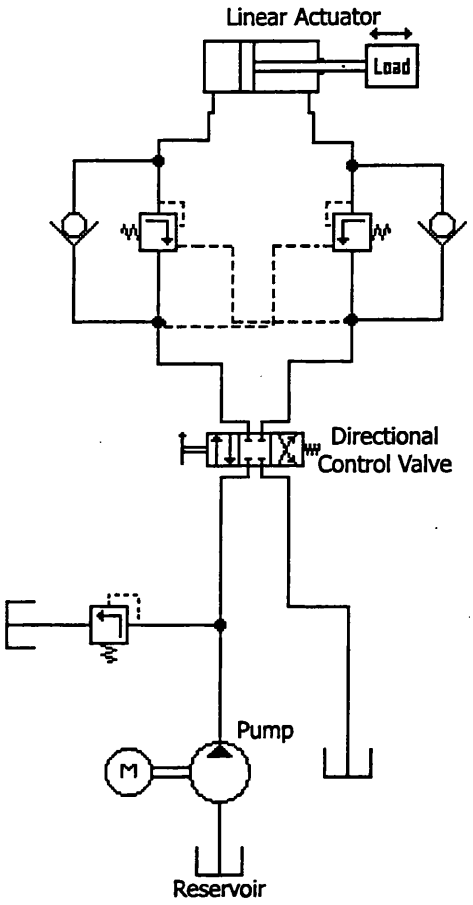


Figure 7. An example fluid power system.

The research reported here concerns the automation of the conceptual design of fluid power systems. To place some constraint on the complexity of the task for the purposes of this research, it was decided to limit consideration to the conceptual design of fluid power systems consisting of a single fluid power *circuit*. Sullivan (1982) makes the following distinction between circuits and systems:

> *"Systems are composed of circuits. A system is capable of completing one or several operations that constitute a work cycle, and includes the pump drive, pump, reservoir, valves, cylinders [i.e., linear actuators], motors, and suitable plumbing to transfer fluid at high pressure. A circuit, on the other hand, is capable of performing one or more specific tasks, but not a complete work cycle."* (p. 295)

So, circuits perform more 'atomic' operations, and systems, as combinations of circuits, perform combinations of these operations. In this respect, the simple system shown in Figure

7 is composed of a single fluid power *circuit*, since, with a single actuator, it performs an elemental operation.

The next section of this chapter contains a more detailed treatment of the process of circuit design, to enable the nature and extent of the conceptual design task to be appreciated. If the task is to be performed by a computer system, some tractable method of representing the 'external' aspects of the problem must be devised. Consequently, the subsequent section deals with the representations that have been developed for depicting design specifications and design solutions in this domain.

In addition to this representational knowledge, an automated system would require heuristic knowledge to transform particular specifications into actual solutions. Since this research is concerned with exploiting the heuristics implicit in examples of design episodes, this chapter concludes with a discussion of an archive of examples of this design task that has been constructed for this purpose.

5.1 The Design of Fluid Power Circuits

In this section, the intention is not to describe *how* fluid power circuits are designed - to do so would require a detailed algorithm, and none is available. Rather, the intention is to identify the activities that are performed during the conceptual design task, in order to gain some appreciation of the scope of the task in this domain. This is a necessary initial step if this task is to be represented successfully in a computer system.

Sullivan provides a general impression of the complete design process:

> *"The design and analysis of hydraulic systems and circuits is systematic in that it considers several activities in sequence to develop and prove the operation of an objective oriented machine. Typically the following steps are followed:*
>
> 1. *Size actuators from output objectives*
> 2. *Establish work cycles using time, flow, pressure, and horsepower plots*
> 3. *Design the circuit*
> 4. *Size and select components*
> 5. *Assemble the circuit or system*
> 6. *Monitor performance of the machine*
> 7. *Check the machine for safe operation and compliance with [health and safety] standards."* (pp. 296-297)

Obviously, this goes beyond the scope of the design process as it is considered here, since the later steps encompass the physical construction and validation of the circuit. Furthermore, there would seem to be something of a conflict with the sequence of design activities in the Pahl and Beitz model, as described in chapter 1. Here, Sullivan's first step would seem to move beyond conceptual design into embodiment, and perhaps even detail, design, to establish very precise information about the circuit's actuator(s). The following steps, however, would seem to conform more readily to the Pahl and Beitz model: step 2 is clarification of the task, step 3 is conceptual design and step 4 embraces both embodiment and detail design.[3] (Sullivan's model of the design process in this domain is, on the whole, corroborated by Henke (1983).)

Considering these third and fourth steps in greater detail, then, the design of fluid power circuits can be seen to involve a number of (not necessarily independent, nor sequential) sub-tasks on the part of the designer, including:

- The selection of hydraulic components to meet the design specification. There are a number of different generic types of component, of which an arbitrary number may be used to solve any given problem.
- The definition of the connections between components, using standard hydraulic pipes. Every component has ports that accept these connections.
- The definition of auxiliary, signal connections in the system. The behaviour of some components is governed by remote signals, in addition to the fluid pressures and flows provided by the hydraulic connections. These signals can be hydraulic in nature (in which case, the signals are conveyed by 'pilot lines') or else electric ('signal lines'). A pilot line will typically be connected to some point in a remote pipe in the system; the pressure at this point provides the signal (the dashed lines in Figure 7 depict pilot lines). An electric signal will typically be produced by some source external to the fluid power system. Every port, hydraulic, pilot and signal, of a component must be connected before the system can be considered complete.
- The definition of the necessary external electrical control elements of the solution. These will provide the necessary electric signals.
- The selection of particular manufacturers' models for each of the selected components, and for each of the pipes, and for the external control elements.

---

[3] Presumably, by "select components" in step 4, Sullivan means "select *manufacturer's models for* components".

- The definition of any operational model parameters that are necessary.

Here, however, interest lies solely in the *conceptual* design of fluid power circuits. As defined in chapter one, the outcome of this task is a scheme design that will achieve all of the principal functionality demanded in the specification. The sub-tasks described above encompass the embodiment and detail design stages, in addition to the conceptual design stage. For the purposes of this work, it is asserted that the principal functionality is expressed in the selection and connection of the generic types of component. In other words, given a specification, the conceptual task is to produce a design scheme something like that shown in the circuit of Figure 7, and expressing a similar degree of information about the solution as this figure does. In other words, when a scheme has been generated, decisions will have been made about the types of generic components that are to be used and their connections, but no precise quantitative values will have been determined. At this point it is considered that decisions will have been made about the means of providing all the principal functionality required. Hence, the conceptual design stage is completed with (roughly) the completion of the first three sub-tasks above (and, implied by this, is that these three sub-tasks can be performed before the others).

The assumption is made, then, that it is possible to move directly from a suitable description of the specification to this qualitative conceptual design, and that the information produced during steps 1 and 2 of Sullivan's methodology is either provided (explicitly or implicitly) in the design specification or else is not needed for the conceptual design task. This is not necessarily a safe assumption, but it is adopted as a matter of expediency for this research.

The supposition that conceptual design is completed with these three sub-tasks is also potentially unsound. Often, for example, the required functionality will be provided by the electrical control acting in tandem with the hydraulic circuit. Sullivan makes the following distinction between *open loop*, *closed loop* and *servo systems*:

> *"Hydraulic circuits may be classified as open loop, closed loop, and servo systems. Open loop systems operate without feedback from the output, except for the operator. Performance is determined by the operational characteristics of the individual components. Most industrial circuits are of this type. Closed loop circuits sample the output and generate a proportional control signal that is used to correct the input command signal...Servo-systems feed back control*

> *signals to the input command as a result of a change in the mechanical position of the output."* (p. 296)[4]

This research at this stage is primarily concerned with the configuration of the fluid power elements, rather than the design of electrical control circuits, so the decision has been made to limit the scope of this work to the consideration of open loop systems. Since the source of the design knowledge is to be a set of example designs, this has been achieved by disregarding example designs in which functionality is rooted in the electrical control elements.

So, the conceptual design of fluid power circuits, for the purposes of this work, is considered to be initiated with a complete description of desired functionality as the design specification. It is completed when descriptions have been devised of one or more complete circuits of connected hydraulic components that, following the selection and sizing of suitable models for the components and connections, will provide the desired functionality. Accordingly, these circuits must be fully connected, incorporating hydraulic control (pilot lines) as necessary. Once this initial configuration has been performed, then it is assumed that all the fundamental decisions about the manner in which the principal functionality is to be achieved will have been made.

5.2 Representation of the Design Task

To describe this class of conceptual design task for an intelligent system, some way must be found of expressing in formal, computationally tractable terms both the design specifications and the design solutions – these are the *representations* of specifications and solutions.

From the discussion of knowledge in chapter 3, it should be apparent that an astute choice of representations is crucial - if the problem is not described in useful or appropriate terms, there is no reason why it should be solvable. Winston (1992) remarks:

> *"Once a problem is described using an appropriate representation, the problem is almost solved."* (p. 18)

This is, perhaps, overstating the case somewhat, but nonetheless it stresses the importance of this stage.

---

[4] It should be noted that Sullivan's comments are nearly twenty years old. The domain technology has advanced, and there is probably a greater proportion of closed loop and servo circuits in use nowadays.

For simple, low-level problems, formal descriptions of the inputs and outputs of a process can readily suggest themselves. Unfortunately, producing this sort of description of more complex problems, such as design tasks, is not a trivial task, and requires a certain amount of knowledge about the problem.

As far as possible, elements of the task should be represented in a manner which eases the process of reasoning with them. This may not correspond to the most immediately available description. For example (and as will be seen later), it may be more appropriate to consider a group of domain components, which together achieve some recognisable elemental function, as a single representational element, rather than as individual component elements. However, this level of representation can rarely be produced without a more than superficial knowledge of the problem.

Different representations, then, can introduce different amounts of task knowledge. A good representation can lessen the burden on the knowledge required for reasoning about the task; on the other hand, a poor representation can render the task virtually impossible. However, this raises the question of the source and the availability of this representation knowledge.

Chapter 4 cited the three ways by which knowledge may be acquired for intelligent computer systems, as identified by Buchanan and Shortliffe (1984). To reiterate, these are:

- *handcrafting* – the representation is devised directly by someone with knowledge of both the task and the requirements for a suitable representation.
- *knowledge engineering* – the consultation of an expert designer to try to access develop a suitable representation.
- *machine learning*.

For the task of devising representations, machine learning algorithms can be discounted: rather than learning representations, these algorithms expect suitable representations to be provided, and will then learn about these.

Both the remaining approaches have their own problems. As described earlier, knowledge engineering techniques can be time-consuming and difficult: however, the problems associated with acquiring representation knowledge in this way would seem to be less serious than those associated with capturing heuristic knowledge. As mentioned earlier, Michie (1982) thinks that experts can provide a list of features which describe a problem, even though they might not be able to describe how to solve the problem.

However, this knowledge engineering approach was ruled out here on the grounds that no suitable expert was available for the length of time that might be needed.

Hence, handcrafting is the remaining approach. Handcrafting the representation would require either an expert designer who is familiar with representation techniques and issues, or else, a system developer who is prepared to learn about the task. Again, the former approach could be discounted due to the lack of a suitable expert. Therefore, the approach taken in this case was one of familiarisation with the task. Potentially, this could be an even lengthier process than knowledge engineering, and, if not done thoroughly, could lead to misconceptions about the task and the introduction of inappropriate representations. However, for want of a viable alternative, this would have to be done. Moreover, it was felt that such a familiarisation would also be useful for understanding and modelling all of the stages in the design process. (It was presumed that the level of proficiency to enable this handcrafting would be somewhere below the level required to actually perform this task, and so could be achieved with the expenditure of a reasonable amount of time and effort.)

So, then, representations would be handcrafted for specifications and solutions, the inputs and outputs of the conceptual design process. These representations would correspond to declarative domain knowledge in the prospective design system, since they indicate the entities which constitute the fluid power domain, at least as far as this task is concerned. This is a necessary step in the development of intelligent systems; the domain knowledge must be explicitly defined, since these are the elements which the procedural knowledge refers to and reasons with.

An added consideration when deciding upon the form these representations should take is the desire to use inductive machine learning techniques. Ideally, the representation of any task should be governed by the nature of the problem itself, and not by some chosen implementation strategy. However, given the desire to investigate design automation, and the bottleneck in acquiring design synthesis knowledge, it would seem perverse to devise some representation that is not compatible with machine learning algorithms.

To be acceptable to the machine learning algorithms, the problem must be described in a consistent manner. As seen in the previous chapter, inductive ML algorithms generally expect examples to be expressed in terms of attribute-value pairs. Each example is described using the same set of attributes. With each attribute is associated a set of values; one value from this set describes the attribute in a given case. This, then, outlines the form which the specification and solution representations are to take.

The handcrafted representations of the design specifications and of the design solutions for the fluid power circuit conceptual design task will now be discussed in turn.

5.2.1 Design Specification Representation

The expression of design specifications, in particular, is fraught with difficulties in most domains. The scope for potential specifications, both in terms of their content and the manner in which they are expressed, is extensive. In general, these difficulties arise because design is fundamentally a human process, with the intention of modifying the environment for some end. As with all communication, expression of the specifications takes place within this context, and assumes shared linguistic, cultural and social references between the customer and the designer. In addition, the specifications are often developed as a result of this interaction between customer and designer. Acquiring the vocabulary and semantics necessary to discuss design problems is itself a complex learning task.

Obviously, these points of reference are unavailable to a computer. When attempting to model design specifications on computer, an attempt must be made to compensate for this lack. The fact that ML algorithms (and intelligent systems in general) expect information in the form of a number of discrete textual symbols or values imposes a further constraint on the form of the representation, since design information is often most conveniently communicated in the form of a sketch or graph.

The desire to use ML algorithms imposes a further limitation on the communication of the specification. To express more, or more complex, functionality, it would seem reasonable to use a larger specification, containing more information, than would be the case for simpler problems. In general, however, the ML algorithms expect all examples to be described using all of a fixed set of attributes. Hence, specifications are of the same 'size', regardless of their complexity.

So, to summarise, the devised representation should allow the communication of sufficient information to produce the conceptual design solution and permit a realistic breadth of expression. Then again, it must also be tightly constrained, in that it defines the totality of the common language between user and the system, preventing the introduction of *ad hoc* terms, which, by definition, would be outside the 'understanding' of the computer. Furthermore, it must be described in terms of a fixed set of discrete symbolic attributes.

To select an appropriate descriptive level for this representation, it is necessary to have some idea of the backgrounds and capabilities of the intended users of the prospective computer system. In this case, it was decided that the user should be familiar with common

engineering notions of mechanical systems, such as *load* and *inertia*, but not necessarily with the domain of fluid power systems. As a consequence, the representation of the specification should omit, as far as is possible, any reference to terms that are specifically related to this domain.

In terms of the content of the specification representation, since this is the description of the conceptual design task, primarily the representation should be focused upon expressing the required functionality, omitting features such as operational or working environment constraints.

Both Sullivan (1982) and Henke (1983) stress the importance of *characterising* the load in fluid power circuit design – in other words, of determining its magnitude, motion, velocity changes, etc. during the operation of the circuit. This characterisation will have a major influence on the final design. While not necessarily characterising the load explicitly (which is part of the design process itself), the specification should contain sufficient information to allow the characterisation to be made. This provides a focus for the descriptive terms used in the representation.

The assumption is also made that the complete specification is available at the outset of the conceptual design phase. This is not always the case, but is assumed here as a matter of practicality.

It should be evident that a great many different representations of the design specification are possible. With the above considerations in mind, two distinct representations have been developed in the course of this research. For their development, an understanding of the design task was acquired, and a study made of the available examples of design specifications in the domain. The first of these representations, termed the *temporal state* representation, describes the functionality of the circuit in terms of a sequence of sets of numerical and qualitative values. For the second representation, the *static state* representation, the functionality is described by the values assigned to a set of purely qualitative attributes.

5.2.2 Temporal State Specification Representation.

It was recognised that the specifications of fluid power circuits are often in the form of a description of the sequence of actions that the circuit must perform (this description may be verbal or graphical). So, it was supposed that changes in the state of the system from one action to the next, and the time taken for such changes to occur, are important to the characterisation the load, and therefore also to the choice of components and their

connection. For instance, a load that is required to change its direction of motion almost instantaneously rather than being gradually slowed and then changing direction, might entail greater pressures within the system and thus a different configuration to cope with these pressures.

In order to capture this information, this concept of defining the design specification in terms of a series of states must in some way be modelled. The representation chosen was to specify a state in terms of values for each of a number of attributes describing the desired output characteristics. An arbitrary number of such state descriptions constitute the desired sequence of actions. The attributes, and the values that they can assume, are as follows:

- *load* - this is simply the magnitude of the load in Newtons that is moved by the circuit in the current state. This is a real-valued attribute.
- *distance* - this is the distance, in metres, through which the load is moved in this state. Again, a real-valued attribute.
- *speed* - the speed, in metres per second, at which the load is moved during this state, a real value.
- *duration* - the duration of this state, in seconds (note that different states in a particular sequence can be of different duration). A real value.
- *direction* - a binary-valued attribute indicating whether the load is being extended or retracted relative to the actuation device in this state. This attribute can assume the value *extension* or else the value *retraction.*
- *plane* - a binary-valued variable indicating whether the motion is occurring in the horizontal plane or a non-horizontal plane in this state. Two values, then, *horizontal* and *non-horizontal.*
- *velocity control* - a binary-valued variable indicating whether or not the solution requires some facility to allow the user to govern the (unspecified a priori) speed during this state. Again, two possible values, *yes* and *no.*

It was recognised that not all this information would be necessary for each state – for instance, if the user was unconcerned with the speed of the load during some state (however, this would later present something of a problem, since in order to use such a specification with ML, it is necessary to have a value for each attribute). It was also recognised that not all of this information is independent - for example, the speed, duration and stroke variables. However, this manner of description was retained as a concession to the different styles in which requirements are provided and the different concerns of the users (an auxiliary check would ensure that nonsensical situations were avoided). The number of states chosen is solely dependent on the number necessary to completely specify the desired behaviour of the

system. Note that in order to describe the relative direction of motion during a state, it is necessary to introduce a fixed reference point. This is the actuator of the circuit itself, and motion is either away from it (*extension*) or towards it (*retraction*).

By way of an example of the use of this representation, Table 3 shows an example design specification. This specification specifies that the resulting system should be able to perform a 3-state horizontal motion. State one involves extending a load of 100000 N a distance of 0.7m in 100s. During state two, the load, now increased to 100500 N, is retracted back over the same distance in 200s. The final state is to once again extend a load of 100000 N over 0.7m, but without the stipulation of a time in which this is to be achieved, but with the requirement for some manner of controlling the speed of motion.

attribute	*state 1 values*	*state 2 values*	*state 3 values*
load	100000	100500	100000
distance	0.7	0.7	0.7
speed	-	-	-
duration	100	200	-
direction	extension	retraction	extension
plane	horizontal	horizontal	horizontal
velocity control	no	no	yes

Table 3. An example design specification, described using the temporal state representation.

This representation has some appealing features, notably the facility to express sequential circuit operations, and its use will be seen in a later chapter. However, this representation presented some difficulties for the ML algorithms. The arbitrary number of states that can be invoked to describe the specification is difficult to translate to the algorithms and means that the apparent consistency of different training examples is reduced. In addition, to describe a training example in this fashion, quite detailed specification information about that example is required, and, in many cases, was found to be lacking. Both these problems will be discussed in greater depth at a later stage. Furthermore, a number of the attributes are clearly not independent of one another. This provides difficulties in that it becomes syntactically possible to specify what are semantically nonsensical problems (for example, specifying that the load is to cover a distance of 1m in 1s at a speed of 2m/s). Most ML algorithms assume independent attributes in their operation. So, while it may be that for complex tasks no guarantee of the complete independence of attributes can be made, this should be attempted as far as possible when producing representations.

In response to these difficulties, a new representation has been devised. This relies on a single, 'static' set of qualitative attribute values to describe the circuit functionality.

5.2.3 Static State Specification Representation

This second representation, then, is composed of a set of 14 attributes. To describe a particular design specification, for each attribute an appropriate choice must be made of a limited number of values. These attributes are as follows:

- *maximum load/force* – the maximum magnitude of the load/force during the operation of the circuit. This attribute can assume one of three values: *low* ($<1x10^4$N), *medium* ($1x10^4$N – $1x10^6$N), *high* ($>1x10^6$N). (These numerical values, and those of the next attribute, were determined by inspection of the figures given in the available examples of the fluid power circuit design, and the recognition that these bands *seemed* to typify circuits. In using these qualitative values, the assertion is made that these 'fuzzy' bands, and not the precise values, impart sufficient information for the designer.)
- *maximum speed* – the maximum magnitude of the speed. Again, described using one of three values: *low* (<0.01m/s), *medium* (0.01m/s – 1.0m/s), *high* (>1.0m/s).
- *plane* – the plane in which the motion occurs (assumed to remain constant throughout the operation). Binary-valued, either *horizontal* or *non-horizontal.*
- *continuously variable speed* – whether or not the speed range to be continuously variable during the motion. In other words, whether the circuit operator is to be provided with a means by which to control the speed between zero and *maximum speed* - values: *no* or *yes.*
- *hold load stationary* – whether or not the facility to hold the load stationary at any position during operation is required. Either *no* or *yes.*
- *smooth accelerations* – whether or not the accelerations and decelerations are required to be (particularly) jolt-free. Either *no* or *yes.*
- *hold load on failure* – whether or not there is a requirement that the load be held stationary in the event of system failure. Either *no* or *yes.*
- *load-independent speed* – whether or not the speed of motion is to be independent of the magnitude of the load. Either *no* or *yes.*
- *control extend speed* – whether the speed of actuator extension is to be controlled (through being set to some pre-defined speed). Either *no* or *yes.*

- *control retract speed* – whether or not the speed of actuator retraction is to be controlled (through being set to some pre-defined speed). Either *no* or *yes*. (Note the distinction between these two attributes and *continuously variable speed*. For the latter, the speed is to be controlled by the operator while the circuit is functioning, whereas, for *control extend speed* and *control retract speed*, a mechanism is to be provided to set the speed 'off-line', that is, between operations of the circuit.)
- *motor required* – whether or not the solution is dependent on a rotary motor. Either *no* or *yes*. (Although this attribute refers to an element of the solution domain, it was felt to be necessary since a number of design examples were encountered in which the decision to use a motor had already been made, and the specification was for a fluid power circuit to drive this.)
- *control inertia* – whether inertial effects within the circuit are to be controlled. Either *no* or *yes*. (This covers a number of different effects, primarily to do with the pressure of the circuit: sometimes, there is a desire *not* to control these, but instead to make use of them in the operation of the circuit.)
- *energy efficiency paramount* – whether or not energy efficiency is to be of paramount importance. Either *no* or *yes*. (This is primarily a trade-off against the cost of a circuit.)
- *control accuracy* – the degree of accuracy that is required in the control of the circuit. Either *low* or *high*.

So, by way of an example, the example specification given above in Table 3 in terms of the temporal state representation might be described in terms of this second representation as shown in Table 4. Note that, for the specification to be considered complete, one of the valid values must be provided for each attribute.

Now that alternative representations have been devised for describing design specifications, attention can turn to the second part of the domain knowledge – the representation of the conceptual design solutions.

5.2.4 Design Solution Representation

The design solutions in this case are qualitative descriptions of fluid power circuits. The representation of these design solutions is, in some senses, a simpler task than the representation of the design specifications, in that descriptions of solutions are explicit, out of necessity, and made up from known elements.

However, the conventional human representation of a conceptual solution in this domain is through the use of a circuit diagram (Figure 7 is an example). These diagrams use a standard set of symbols for representing the components and the connections that exist between them. Although this representation is a formal one, it is not compatible with the attribute-value form generally expected by the ML algorithms. It must be re-described to make it comply with this form, and in such a manner as to express all of the essential information conveyed by the diagram.

attribute name	*value*
maximum load/force	medium
maximum speed	low
plane of motion	horizontal
continuously variable speed	yes
hold load stationary	low
smooth accelerations	no
hold load on failure	no
load-independent speed	yes
control extend speed	no
control retract speed	yes
solution requires motor	yes
control inertia	no
energy efficiency paramount	yes
control accuracy	no

Table 4. The same example specification, as described using the static state representation.

An additional difficulty arises because, although there is only a certain number of *types* of component, an unlimited number of components of a particular type could be used, and so there is no theoretical limit to the size (in terms of numbers of components and their connections) of design solutions. ML algorithms, though, generally require representations of a constrained and consistent size. A further consideration, as mentioned in section 5.2, is the desire to represent the elements of the solution at a level which facilitates reasoning with them.

Through familiarisation with the task and the domain, it was recognised that, on the whole, conceptual circuit solutions in this domain share a similar basic framework. This feature of solutions was exploited as a representational aid in the form of the solution *template* (see Figure 8). In representational terms, the template performs a twofold function. First, it supplies the basic functionality integral to all solutions (there is a power source in the form of a basic pump, an actuator for supplying the energy, and the facility to direct flow from pump to actuator by means of a directional valve and hydraulic pipe-work). Secondly, it provides a skeleton configuration, and additional components can be inserted into a number of labelled *slots* so as to provide the particular functionality demanded by the specification. In this manner, the template offers a convenient context for describing complete circuit solutions: a solution consists of the template along with the additional components, their corresponding slot positions and their 'orientations'[5] in those slots. Any slots without components can be considered filled by a direct pipe connection across the slot. Note that the pump, actuator and directional control valve in the template are also labelled as slots; this is to permit the replacement of the standard types of these components by more 'sophisticated' models, which are able to provide some particular additional functionality.

It should be evident that these slot labels need to be used with consistency, both in describing existing design examples and when generating new solutions, otherwise regularities in the domain might go unrecognised and confusion may occur. Slot C is relative to the pump, and slot D always occurs in the line returning flow from the directional control valve to the reservoir, so there should be no difficulty identifying these slots in any example. Likewise, the pump, actuator and directional control valve slots are fixed to these components. Problems do arise, however, with the A and B slots. These would appear to be labelled arbitrarily and any components placed in them would appear to be interchangeable. Indeed this is the case, except in two particular sets of circumstances:

- when the system is operating against an off-horizontal load, component(s) will often be placed in one of these slots (that slot in the pipe leading to the side of the actuator acting against gravity) to counteract the gravitational effects in the system.
- when some function is asked for which is relative to the actuator (for instance, *control the speed of actuator extension*), then the chosen component(s) will need to be placed in one or other of the slots relative to the actuator to provide this function.

---

[5] 'Orientation' here refers to the manner in which the ports of the component are connected to the 'ports' of the slot. This idea will hopefully become clearer later on.

To deal with these situations, the following convention is needed:

> *"the piston side of the actuator is considered always to 'act' against gravity in off-horizontal circuits. Slot A always occurs in the pipe between the actuator piston port and the directional control valve."*

(The piston port is that labelled '1' of slot *ACT* in Figure 8.) This convention is used when interpreting examples to decide which slot a component is in. In circuits having a motor rather than a linear actuator, the piston-side equivalent of the motor can usually be recognised.

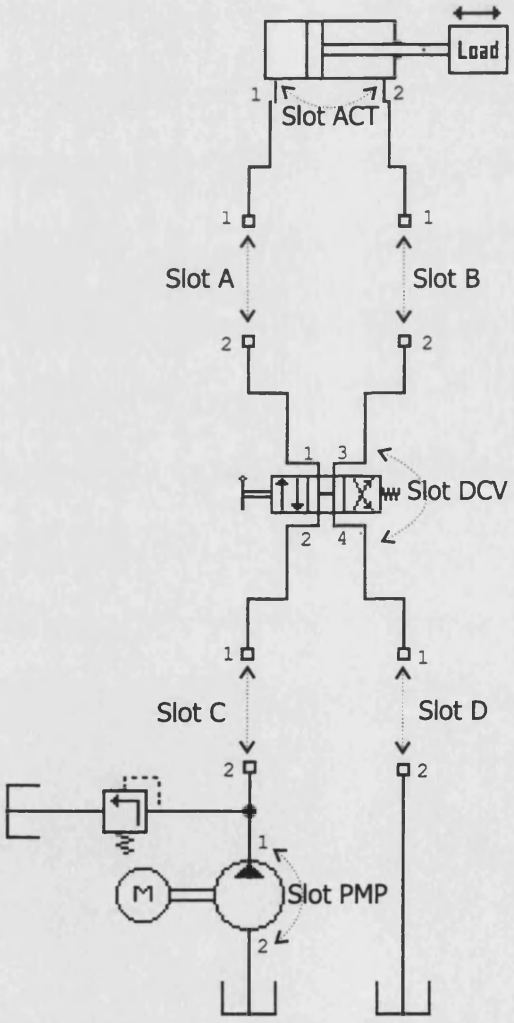


Figure 8. The solution template.

The next step, then, is to define the total set of additional components that are required to describe solutions, and identify the positions and orientations in which they are used. While in theory any domain component can be placed in any slot with any orientation, in practice only certain components used in certain ways perform useful functions; only these useful elements, then, are necessary for describing valid solutions. In addition, certain groups of components, which occur frequently in solutions arranged in some consistent manner, are considered to perform 'atomic' functions; it is conjectured that each of these groups is reasoned about as a single entity, and, as a consequence, that is how each is represented.

So, solutions can be represented as the combination of template and a number of *solution elements*. Each solution element consists of:

- one or more components, arranged in a particular manner, with any hydraulic pipe work necessary for connecting components together and to the template;
- the label of the template slot associated with the element, and;
- some indication of the particular orientation of the element within that slot.

There is no need to have more than one particular element in a solution, since this would denote an unnecessary duplication of function. Different solution elements occupying the same slot are considered to occur in (arbitrary) series in that slot.

In addition, as discussed earlier in this chapter, a number of components have ports that require a pilot line connection. A pilot line supplies an indication of the pressure at some remote part of the circuit, and since the operation of these components is dependent on this, the source of this connection must be specified to give a complete circuit. In general, these pilot signals are taken from hydraulic pipes, and so, in theory, could be drawn from any point in the circuit. However, consideration of example circuits indicates that the pilot signals associated with particular elements tended to be supplied from the same relative positions, which can be described as being a 'port' of one of the other slots. This allows a simple 'heuristic' to be added to complete the element description, indicating the source of the signal, and hence, the point in the circuit to connect to the element with a pilot line. So, there is a fourth constituent of each solution element:

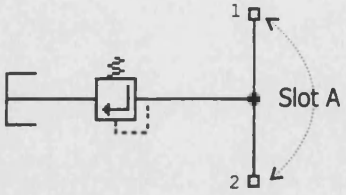
- possibly some 'heuristic' indicating the source of the remote pilot signal necessary to the element's behaviour.

In total, 17 different solution elements are proposed as being sufficient to describe solution circuits for this conceptual design task. These elements were determined by examination of

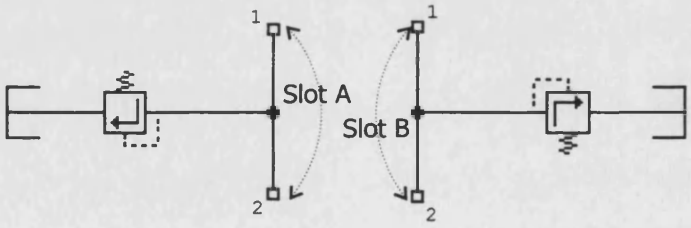
the available design solutions. So, with the template considered common to all solutions of this type, 17 attributes, one corresponding to each element, describe the range of possible design solutions. Each attribute can assume the value *present* or else *absent*. To completely specify some particular solution, one of these values must be associated with every attribute.

These 17 recognised solution elements are as follows (note that the numbering of the slot 'ports', with concordance to the similar numbering in the template, gives the necessary orientation information; also note the pilot line 'heuristics' associated with some elements):

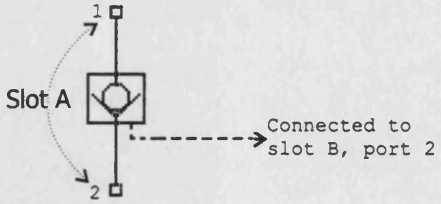
- pressure relief valve 1 (template slot A) (code: PRV1_A):



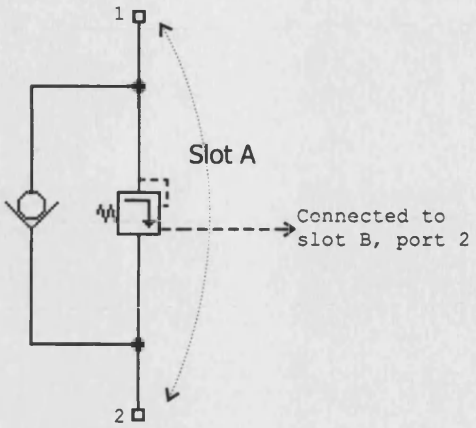
- pressure relief valve 1 (A and B) (code: PRV1_A&B):



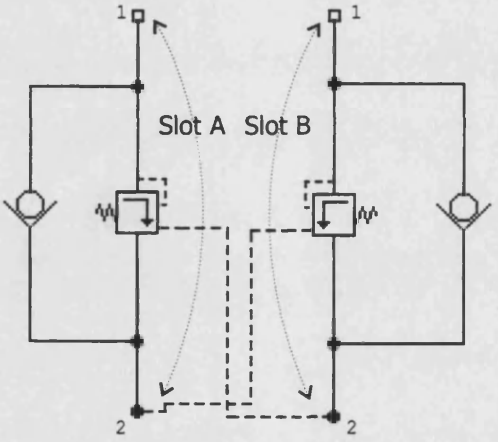
- pilot-operated check valve (A) (code: POCV_A):



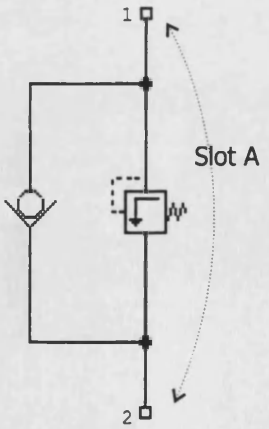
- counter-balance valve 1 (A) (code: CBV1_A):



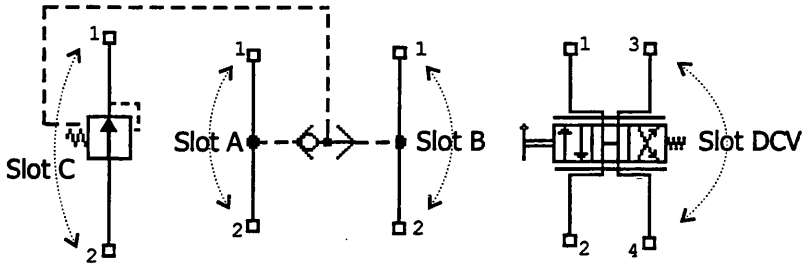
- counter-balance valve 1 (A and B) (code: CBV1_A&B):



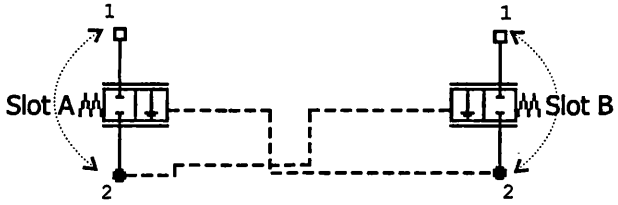
- counter-balance valve 2 (A) (code: CBV2_A):



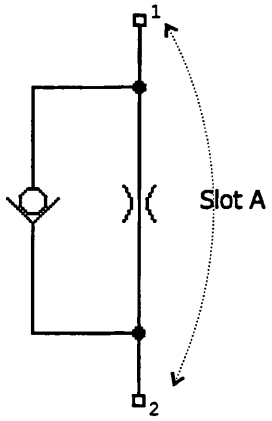
- pressure compensator (C) and proportional valve (DCV)
  (code: PCMP_C&PROP_DCV):



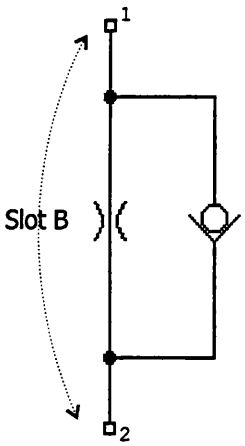
- deceleration valve (A and B) (code: DECV_A&B):



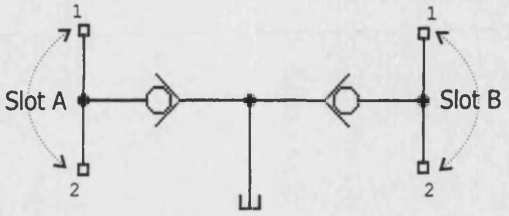
- meter-out valve (A) (code: MO_A):



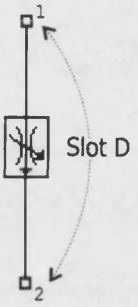
- meter-out valve (B) (code: MO_B):



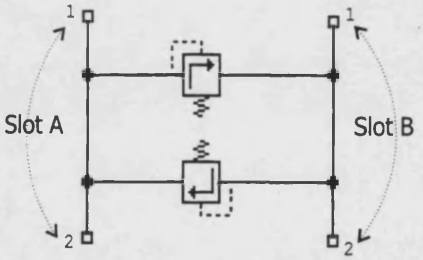
- check valve combination (A and B) (code: CVC_A&B):



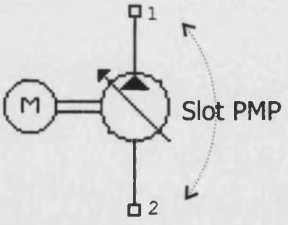
- variable pressure-compensated restrictor valve (D) (code: VPCRV_D):



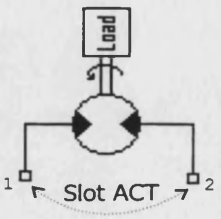
- pressure relief valve 2 (A and B) (code: PRV2_A&B):



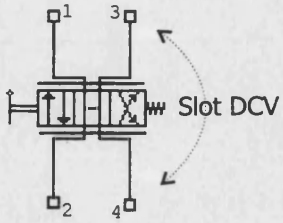
- variable displacement pump (PMP) (code: VDP_PMP):



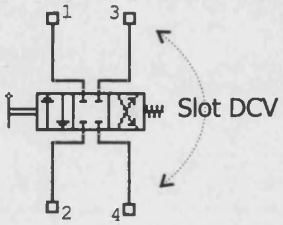
- motor (ACT) (code: MOT_ACT):



- proportional valve (DCV) (code: PROP_DCV):



- closed-centre spool (DCV) (code: CC_DCV):



As with the specification representations, this description introduces particular declarative domain knowledge into the system. It is important to note that the solution elements are at a higher conceptual level that that of the basic domain components: an element includes pipe work, and relative position and orientation information. In addition to providing a tractable representation, it is asserted that this combination of information provides a better foundation for reasoning about the problem. Whereas a component, on its own, suggests only a behaviour, a solution element, having a relative position in a circuit, suggests a particular function. This functional level of representation would seem to be more appropriate for conceptual design.

There remain a number of problems with this representation, however. Although only a single template is used here, it is an over-simplification to consider that all circuits have this as their basis. For example, hydrostatic circuits would seem to require a quite different 'closed-loop' template. Furthermore, it is not immediately obvious how this template approach could be scaled in the future so as to reason about *systems* of circuits.

In addition, there are probably more than the 17 solution elements described here. However, this represents the sum of the elements encountered in the examples, and, since these examples form the basis of the heuristic synthesis knowledge, these are the only elements to which this knowledge can refer.

Finally, the relative positions of several elements occupying the same slot is not handled. Presumably this ordering will have some bearing on the functionality of these elements.

5.2.5 Representation of the Design Problem — Issues

From the above discussion, it should be evident that the creation of representations of design problems for intelligent computer systems involves a certain amount of pragmatism, there being no 'right' representations, merely more useful ones in particular contexts. It seems worthwhile summarising a number of the general issues that surround the representation of design problems:

- The development of useful representations requires a certain amount of knowledge about the design task - the definition of representations is itself a knowledge acquisition task. Currently there are two practical approaches available, namely, knowledge engineering and handcrafting. The problems that can arise during knowledge engineering have been discussed in chapter 3. Handcrafting, on the other hand, requires either a design expert who is familiar with representation techniques, or else that the intelligent system builder becomes familiar with the domain and task. The former seems unlikely, and the latter difficult and time-consuming. So, neither the knowledge engineering nor the handcrafting approach offers an easy route to good representations.

- To be useful to any intelligent system (and especially so if ML is to be applied), the representations have to constrained, to give a degree of 'closure' to the more open world of terms and references in which human designers operate. Often it may be necessary to focus on a restricted domain of problems (as is the case here, with consideration of circuit, rather than full system, design), or by focusing on a particular feature of the task. Consequently, representations of design tasks are, at best, some approximation to the 'real' description — the goal must be to ensure that they contain those elements most influential in and relevant to the design process, whilst omitting irrelevant detail. The representation of specifications can be particularly difficult, involving the restriction of theoretically unconstrained expressiveness into a limited set of terms. The translation of the task into the symbolic attribute-value terms expected for most intelligent systems can seem unnatural where information is usually presented in the form of, say, a diagram.

- To fix the level of the representation of, in particular, design specifications, it is important to have some idea of the capabilities and background of the prospective user of the intelligent system. The representation can then be tailored toward this user, through the introduction of attributes at an appropriate descriptive level.

- To ease later reasoning, it might be appropriate to describe constituents of the domain at a 'reasoning' level, that is, a level at which the entities can be manipulated during the design process. If done successfully, this can reduce the burden on the procedural knowledge, but, again, it requires deeper knowledge of the task.

- The desire to use the developed representations in ML contexts introduces additional constraints on their development. In general, ML algorithms expect their learning tasks to be stated in terms of attribute-value patterns, of limited length and expressed in a consistent manner.

5.3 The Source of Knowledge — the Archive of Example Designs

The objective of this research is to investigate ways in which the knowledge bottleneck in intelligent design system development can be overcome through the exploitation of heuristic knowledge implicit in design examples. Obviously, before this can be done, it is necessary to collect suitable examples and describe them in a suitable manner. These examples together constitute the design *archive* for this task. To create this archive, it is necessary to have a clear idea of its purpose to identify the information that each example within it should contain.

In mechanical engineering design contexts, a great deal of information and documentation is generated, from the rough initial design briefs through to construction, operation and maintenance procedures for the completed artefact. In this research, the primary concern is with the conceptual design phase of fluid power circuit design. The nature and extent of this task was defined above in section 5.1. The intention is to apply machine learning to acquire the procedural knowledge required for performing this task, in other words, for translating a design specification into a design solution. The specification and the corresponding solution, then, are the information that is of principal interest, and so, constitute the examples in the archive. This data must be recorded in some appropriate, consistent manner. The specification representations discussed above provide a basis for producing a formal record of that information. The circuit solutions themselves can be stored in the conventional schematic manner (which can be translated into the solution representation given above with relative ease). (It should be noted here that the development of the representations did not strictly precede the creation of the archive; rather, the representations were developed using the available examples. If an example could not be adequately described using the current representations, this indicated that the representations needed to be amended to incorporate this example.)

The examples in the archive come from a number of sources: some are from the design records of companies, some from training material and others from textbooks. Whatever their origin, the assumption is made that each example is of a successful design case. It is

important to realise that, during the research the archive was not static, but evolved, as more examples or information became available, or the course of the investigation caused the emphasis placed on the recorded items to shift. However, two milestone versions exist (Court and Potter, 1996; Darlington and Potter, 1998). The earlier of these includes circuit descriptions based upon the temporal state specification representation, while the later uses the static state representation (and includes more examples). Appendix A provides some details of the content of the second version of the archive.

In addition to the 'real' examples, the archives include a number of *test* examples, created as a benchmark set to compare the intelligent systems constructed.

This archive, then, constitutes the source of heuristic synthesis knowledge for this design task. As such, and as will be seen in the following chapters, it forms the basis of the training data supplied to ML algorithms applied to learning these heuristics - and also the case base for a CBR system. Following a discussion of the issues that are raised through the construction of such an archive, an example will be given of the process by which an example is incorporated into the archive.

5.3.1 Archive of Design Examples — Issues

The development of the archive, the nature of its content, and its intended use, raise a number of important issues:

The Number of Examples

Given that this design task is a relatively complicated one, then it would seem reasonable to expect that the synthesis knowledge necessary to perform it is also complex. Hence, if it were to be extracted from design examples, there would seem to be a need for a large archive to illustrate the full intricacy of the knowledge. However, readily available examples, sufficiently well documented for inclusion in the archive, were found to be relatively few in number (the first archive contains 16 examples, the second 30). It would appear that organisations place restrictions on the release of design data, for whatever reason, and, moreover, specialise in a particular area, and so do not possess a wide range of sample designs.

It might be thought that the restriction of consideration to circuit design based on a single template exacerbates this problem. Indeed, consideration of the complete fluid power systems design task would make available further examples. However, to perform this

extended task, more (and perhaps more complex) knowledge would be necessary – requiring an even greater number of examples.

So, the small number of available examples was felt to provide an accurate reflection of the situation — a lack of actual design data of this form is a fact of life when working in real design domains. Any approach to acquiring synthesis knowledge from examples must be able to work within these constraints: while approaches or algorithms may potentially work well given a large number of data, if these data are simply not available, and never will be, these are not useful approaches.

The Representativeness of the Archive

If accurate, generalised design knowledge is to be learned, so that a wide range of design problems can be solved, it is vital that the archive examples, as a whole, are representative of the task. In other words, the examples should illustrate the use of specifications and the solution elements sufficiently to allow the relationships between the two to be inferred. Unfortunately, there is no easy method of ascertaining the representativeness of a body of data. While no guarantee, a large data set is often used with ML approaches to try to ensure this.

The Description of the Examples

Since design specifications are chiefly stated in natural language, they must be re-described to conform to the adopted formal description. Similarly, the design solutions must be restated according to the adopted representation. In both cases, this involves interpretation of the meaning of the design; misinterpretation can introduce errors into the data, and information can be lost (or added).

The Incompleteness of the Examples

Many examples are incompletely recorded. Chiefly, the problem lies in the description of the specifications, as solutions tend to be well described (of necessity, since they have to be physically constructed at some stage). In addition to poor documentation practices, this incompleteness might be due to assumptions on the part of designers about what needs to be said and what can be inferred from the context of the current problem. Whereas, in natural communication, attributes of the specification that must be satisfied are (or should be) stated explicitly, nothing need be said about those that are *not* demanded. However, given the

incompleteness of the documentation, it may sometimes be the case that attributes *are* demanded yet go unrecorded.

Hence, in a number of cases it has been necessary to manually 'reverse-engineer' some of the specification values. In other words, based on the solution description, assumptions are made about the functionality of the system, and, hence, about the original specification. While increasing the amount of useful data, this obviously increases the likelihood of errors of misinterpretation being introduced into the archive. The reverse-engineering of real-valued attributes, such as the magnitude of the load/force presents a real problem, though. This is a major limitation with the temporal state specification representation, which requires specific values to be documented. It is often easier to supply a range of values for which the solution may be appropriate than to try to supply a precise value. This was one of the factors that led to the development of the second representation for specifications.

The Quality of the Examples

For the examples to be useful, they are assumed to be examples of good design.[6] However, this notion of goodness is subjective. For example, different companies may apply different criteria of differing rigour when judging if a solution meets the given specification. In addition, the circuits built from the designs operate over some time-scale. What may initially seem to be a good solution may suffer some catastrophic failure after a number of years' service due to weaknesses in the initial design.

Aside from these considerations, whenever dealing with real-world data, some allowance must be made for 'noise' (that is, incorrectly recorded or otherwise corrupted data values) in the examples. This problem is exacerbated by the need to interpret the data according to the formal representations adopted, and the reverse-engineering of incomplete examples.

The Expression of the Examples

A further issue arises due to the human nature of expressing design specifications, touched upon above. In general, those attributes which are expressly required in a solution will be stated as such. On the other hand, it has been found that there is a tendency to say nothing in the specification about those attributes which are not required. This situation is subject to

---

[6] Potentially, there is much information to be gained from examples of bad design. However, there is, understandably, a general reluctance to make such information available, as well as a difficulty in deciding what it is exactly that constitutes an example of bad design.

two alternative interpretations. First, this might mean that the attribute is definitely not needed: if a solution does provide the associated function, then it is not a correct solution. Alternatively, this might mean that the customer *doesn't care* if that attribute is satisfied. That is, it is not vital that the attribute be *not* satisfied in the solution; if the solution does so (and presuming that the customer incurs no extra expense as a result), then this does not necessarily invalidate the solution. Again, this is a manifestation of the nuances of human communication.

However, when trying to use this information to learn design synthesis, this loss of distinction in the different interpretations of such 'negative' attributes becomes problematic. Consider the case of a single solution element that satisfies two particular functional attributes of the specification. In some examples, the solution contains this element, and both of these attributes are explicitly demanded. This is fine: the element appears to conform to its functionality. However, in other examples, also with solutions containing the element, one or other of the attributes may not be explicitly asked for, although the solution satisfies them. This is the case in which the customer 'doesn't care' whether the attribute is satisfied or not; it so happens that the produced solution does satisfy it. Now, learning design synthesis knowledge involves, at some level, making some association between attributes of the specification and the elements of the solution that achieve them. That is, it involves recognising those elements that satisfy particular attributes. If, in this case, the 'don't care' interpretation is represented by the value 'no', then the causal relationship between solution element and attribute appears to have been severed (although the attribute *is* satisfied in actuality).

A related problem arises due to the derivative nature of much design - new designs are often formed by modifying existing solutions. However, this can lead to the retention of inappropriate elements in the new design. While these elements do not provide any undesired functionality, they are unnecessary for meeting the given specification: the solution is 'over-engineered'. The acquisition of design knowledge involves learning the reasons for the appearance of each element in a solution; since over-engineered elements have no real functional reason for being in a solution, this can lead to the formation of spurious associations.

There are two possible approaches to maintaining the necessary causality, both of which involve further reverse-engineering of values. First, an additional value might be introduced for each relevant attribute of the specification, allowing the explicit expression of 'don't care'. Alternatively, the specification for each example could be re-described to reflect the *maximal* functionality that the circuit provides. In other words, the values of the

specification attributes are altered to reflect the *actual* functionality that the solution gives, rather than that subset which was explicitly demanded. This latter approach has been that adopted here.

The Style of Design Solutions

A further issue that crops up when attempting to amass an archive of design examples for learning purposes is that of the *style* of solutions. This problem is most apparent when two or more different design solutions appear to satisfy the same specification. This may be because the representation of the design specification is not sufficiently expressive to capture the distinction that has resulted in the different designs. On the other hand, it may simply be a reflection of the fact that, in complex domains, it is possible to achieve the same effect in a number of different ways. Whatever the reason, it poses a problem for machine learning in that there appears to be a contradiction in the examples: a design specification is associated with two or more solutions — which association is to be learned?

A further stylistic problem arises due to the fact that the example designs collected in the archive have been produced over a number of years. In that time, the nature of mechanical engineering design problems can and does alter: for example, new, more advanced components may become available, legislation introduced, or working practices changed, all of which may change the nature of the designs that are produced. This 'evolution' of the domain may be reflected in an apparent lack of consistency or regularity in designs.

Unlike humans, current ML algorithms are generally not flexible enough to cope with seeming contradictions of this sort. Different algorithms respond to their occurrence in different ways. Some can process this information as noise within the system; others may not be able to process it at all and fail to learn anything of use. Even assuming that obviously contradictory examples are purged from the archive, the problem remains in that differing styles may manifest themselves as very different solutions to quite similar specifications. Such phenomena in the training data can severely restrict the ability of the machine learning algorithm to generalise over the data, limiting the potential of the learned knowledge to reason successfully about problems other than those in the archive.

Type of Design

Design solutions can be classified as being *routine*, *innovative* or *creative* (Gero, 1990). Routine designs use solution elements in a conventional manner, innovative designs use elements in an unconventional manner, and creative designs introduce new elements to solve

the problem. Since the machine learning task is one of learning regularities amongst the design examples, only routine designs would seem to be useful for this task, inasmuch as they use solution elements in a regular manner. Indeed, innovative or creative designs may appear inconsistent with the other examples, or may not even be describable in the same terms.

5.3.2 The Incorporation of an Example into the Archive

This section describes the manner in which a typical example is included in the archive. Figure 9 shows one of the simpler design examples that have been collected into the archive. The source of this example is one of the Fluid Power Centre Courses (FPC, 1997) run by the Fluid Power Centre at the University of Bath.

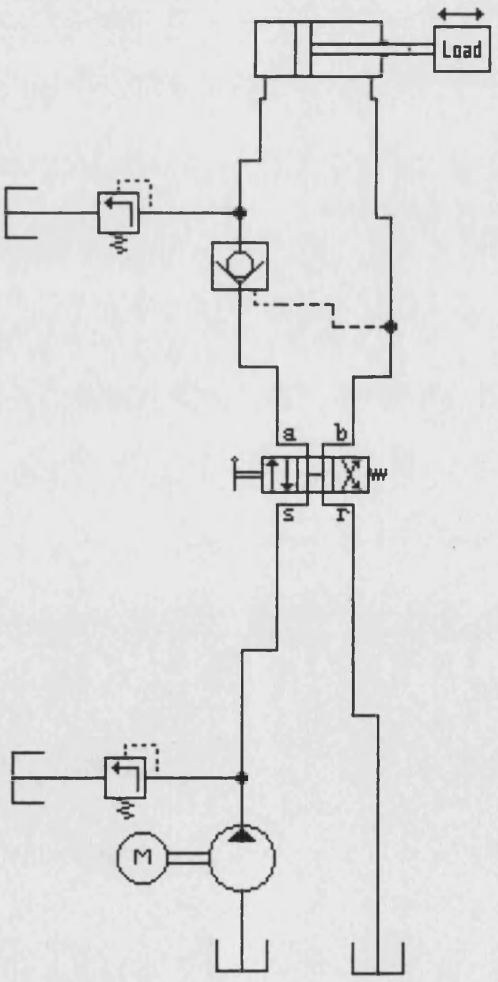


Figure 9. A circuit solution from an archive example.

The specification of this design is given in natural language, and is as follows:

> *"[A circuit is required to] move a mass of 890kg vertically up and down over a distance of 0.61m. It is required to stop the mass at any position within the stroke without excessive pressure peaks and avoiding cavitation during any overrun. When held stationary, the mass must not be allowed to creep downwards."*

This specification is quite sparse and discusses the task at a low level, referring to domain concepts such as *stroke* (the distance through which the load is moved) and *cavitation* (the situation which occurs when a load is moving so quickly that insufficient fluid enters the actuator, which can damage components and cause unpredictable system behaviour). The source of the example probably has a bearing on the character of this description.

So, interpreting this specification in terms of the temporal state representation, the motion is considered to consist of two sequential states (Table 5). Note that some of the difficulties of using this representation are apparent already. The interpretation of the motion as consisting of two states seems too rigid when the verbal specification talks of being able to 'stop the mass at any position during' the motion – indeed, there is no way of expressing this functionality with this representation.

attribute	*state 1 values*	*state 2 values*
load [mass × 9.8]	8722	8722
distance	0.61	0.61
speed	-	-
duration	-	-
direction	extension	retraction
plane	non-horizontal	non-horizontal
velocity control	no	no

Table 5. Example specification interpreted according to the temporal state representation.

When describing these requirements using the static state representation B (Table 6), some degree of interpretation of the solution is required to complete the specification. A value needs to be supplied to the *maximum speed* attribute. Based on the circuit (and in ignorance of the actual value), it is given a value of *medium*. Nothing in the specification, or in the circuit, refers to the need for *continuously variable speed*, *smooth accelerations*, *load independent speed*, *control extend speed* or *control retract speed*, so these are all given the

value *no*. The solution clearly has a linear actuator, so the solution does not require a motor. Again, no mention is made of energy efficiency being a chief concern, nor of the need for high control accuracy, and, since nothing in the circuit suggests that these attributes have been provided, they are given values of *no* and *low* respectively.

The specification *does* demand *hold load stationary*, though. In addition, the manner in which this is achieved in the solution is considered also to *hold load on failure*; so, remembering the desire to express the maximal functionality of solutions, this attribute is given the value *yes*, in spite of no mention being made of this in the specification.

The attribute *inertia control* is also given the value *yes*, since the specification explicitly mentions the need to avoid pressure peaks and cavitation: both these problems are seen as the products of insufficient control of the inertia of the load.

attribute name	***value***
maximum load/force	low
maximum speed	medium
plane of motion	non-horizontal
continuously variable speed	no
hold load stationary	yes
smooth accelerations	no
hold load on failure	yes
load-independent speed	no
control extend speed	no
control retract speed	no
solution requires motor	no
control inertia	yes
energy efficiency paramount	no
control accuracy	low

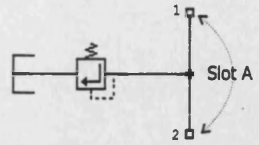
Table 6. Example specification interpreted according to the static state representation.

The next task is to interpret the solution according to the adopted representation. The solution conforms well to the template (Figure 10). In addition, there are two solution

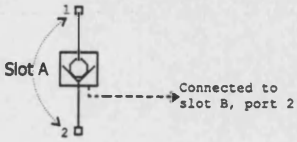
elements, both occurring in slot A (since they are acting counter to gravity in this non-horizontal system).

These solution elements are:

PRV1_A:



POCV_A:



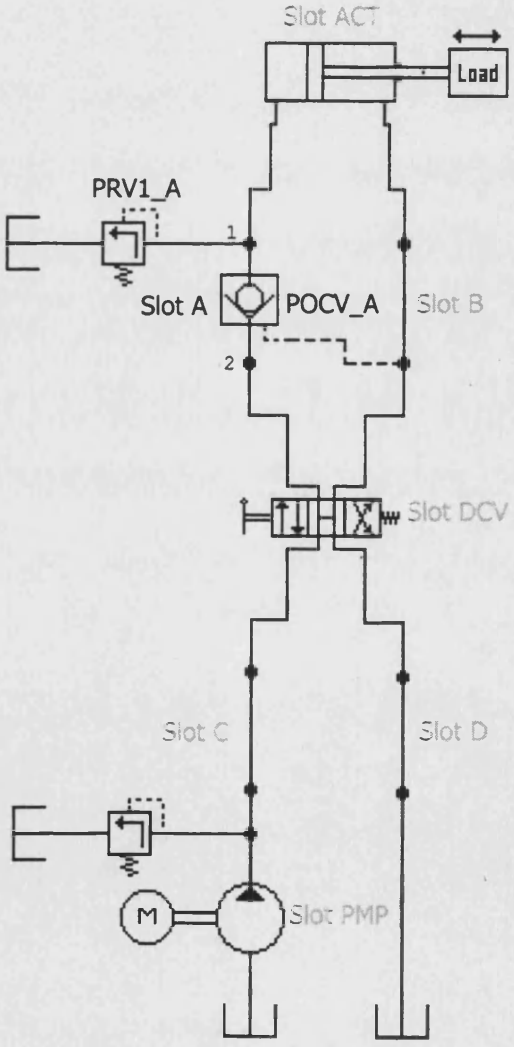


Figure 10. An example circuit.

This example should give some flavour of the amount of creative interpretation that is required to describe each example in the archive. Rarely do specifications contain more useful information than in this case; frequently they contain less. This circuit solution is relatively straightforward to understand, and conforms well to the template; other circuits can be much more complex and less 'well-behaved'.

5.4 Summary

This chapter has described in detail the fluid power system design problem, and discussed the extent of the conceptual design task within this overall design problem. In order to reason about this task within an intelligent system, it is necessary to represent this task in some form to the machine. Concentrating on the functionality of fluid power circuits, two distinct forms of representing design specifications and one of design solutions have been developed. These representations have been described in this chapter, along with some of the issues raised during the process of their development.

A second element necessary for the approach adopted in this research is an archive of design examples. Each example in this archive comprises a design specification and the corresponding design solution. This archive has been developed using material from industrial and educational sources. However, the nature of the design task and the intention to use the archive to learn design synthesis knowledge raises a number of important issues about the expression and adequacy of the constructed archive.

The following chapters discuss the use of these representations and the archive in the production of an intelligent system for the conceptual design of fluid power circuits.

6 Machine Learning and Design Heuristics

Chapter 2 described how the synthesis of conceptual designs for configuration problems depends upon abductive reasoning on the part of the designer. During this form of reasoning, some cause is postulated to account for an effect. In configuration terms, features of a solution are viewed as 'causing' features of the specification. Designers must generate that complete solution which, in their opinion, is 'most likely' to cause the whole specification to be met. Their knowledge of the task allows the designers to make judgements of the likelihood of potential causes; most successful are those designers whose judgements most closely and most often correspond to the reality of the domain.

So, abductive reasoning is subjective – one designer's idea of the best solution to a particular specification may not match that of a second designer. The ability to make successful design decisions is a product of the designer's experience. For the purposes of this work, the assertion is made that this experience is embodied in the form of heuristic knowledge, the generalised 'rules of thumb' that permit a sensible response to new design problems. Chapter 3 discussed heuristic knowledge and Knowledge-Based Systems, computer systems that try to reason using this knowledge – and also, the difficulties that surround any knowledge engineering attempt to acquire the heuristics for these systems directly from human experts.

Inductive machine learning, as discussed in chapter 4, has been proposed as one way of avoiding this bottleneck in the acquisition of heuristic knowledge. If the heuristics are considered to be expressed implicitly in the solutions generated by experts in response to specifications, then, given enough suitable examples of specifications and their corresponding solutions, an inductive ML algorithm may be able to extract this knowledge into a generalised form. This knowledge could then be applied by a KBS to solve other problems of a similar type.

A substantial part of the research reported in this thesis has been devoted to investigating the suitability of inductive ML for capturing the heuristics necessary for the conceptual design of fluid power circuits. Several different algorithms have been applied and their results have been used to construct KBSs for this particular design task. Subsequent chapters describe

some of these applications in detail. First, however, this chapter is devoted to describing the general rationale underlying the attempt to acquire design heuristics in this manner, and the method by which the KBSs are built.

6.1 Knowledge-Based Systems for Design Synthesis

As seen in chapter 3, KBSs generally reason by using their knowledge in a deductive manner: the heuristics are assumed to be valid, and, as a consequence, the conclusions that are drawn using them are assumed to be true. This corresponds to a 'probable deduction' approach to abductive reasoning, an idea introduced in the second chapter - the KBS is presented with a design specification, and from this deduces a design solution using its knowledge base. The success of this approach depends on the quality of the heuristics. Successful abduction, as characterised in chapter 2, involves being able to use the available and relevant information to influence and inform the decisions that are made. Accordingly, if the design heuristics are sophisticated enough to incorporate this information and permit apt decisions to be made in a variety of contexts, then probable deduction will probably result in a decent solution. If, on the other hand, the heuristics merely suggest simple associations, and fail to include the contextual information which influences decisions, then it is likely that this approach will fail.

It is crucial, then, that a design KBS contains the 'right' heuristics for the task – they must express the appropriate knowledge and in a manner which enables them to be properly applied. As has been seen, attempts to capture the heuristics by knowledge engineering can produce less than satisfactory results. So, and as will be described in this and subsequent chapters, inductive ML has been applied to acquiring this heuristic knowledge. Once acquired, the knowledge can be used to construct KBSs for the design task in question. These systems can then be tested to obtain some indication of the quality of the learned heuristics.

6.1.1 Building a Design KBS – the Sources of Knowledge

Section 3.3 described the different forms of knowledge that are required for any design system. To summarise briefly, these forms are:

- *domain knowledge* – describes the entities that constitute the domain.
- *inference knowledge* – describes design decisions that can be made, in terms of the domain entities.

- *strategic knowledge* – describes methods of organising inference knowledge so that the design task can be performed in its entirety.

- *working knowledge* – describes the current design specification, the intermediate inferences made during the process, and, eventually, the design solution.

If a KBS is to be constructed for this particular design task, then the source, form and content of each category of knowledge must be explicitly defined. The following chapters discuss a number of different KBSs; the form and content of the knowledge in these differs from system to system. However, in general, for every system, each category of knowledge has a common *source*, as will now be described.

As explained in the previous chapter, representations have been handcrafted for the design specifications and solutions. This is domain knowledge in all the systems. In addition, one of the systems developed uses an explicit *functional reasoning* approach – and, as a consequence, for this system additional domain knowledge is introduced to allow the expression and use of these functions, as is explained in section 7.3.3. Moreover, certain strategies used in the KBSs decompose the design task into a number of lower-level selections amongst a subset of solution elements based on a relevant subset of specification attributes. This association of relevant subsets also constitutes domain knowledge in these systems.

This additional knowledge has also been handcrafted. As might be expected, in all systems the domain knowledge terms are also used to describe the current state of the working knowledge for a particular design episode.

The inference knowledge and strategic knowledge together constitute heuristic knowledge in the system. The research described in this thesis involves investigating the extent to which 'good' heuristic knowledge can be acquired from examples of previous design episodes. As described in the preceding chapter, an archive has been constructed, consisting of a number of design examples. Each example consists of some specification and the corresponding solution. As such, the examples might implicitly contain the heuristics that govern the choice of solution elements in response to certain values of the specification attributes.

However, the examples appear to contain no information about the order in which these decisions were made, or the precise steps that were followed to generate the solution. Hence, this archive would seem to represent a potential source of *inference knowledge* but not of *strategic knowledge*. (Potentially, additional information could have been used to describe

each design example in greater detail in an attempt to exemplify these strategic decisions, but such information does not seem to be readily available.)

In order to complete the knowledge of the KBS, then, this strategic knowledge would have to be acquired from a source other than machine learning. However, since strategic knowledge is heuristic in nature, and given the difficulties of acquiring heuristics by a process of knowledge engineering, this presents something of a problem. In practice, as will be seen in subsequent chapters, for the most part simple strategies (one per system) have been suggested by the nature of the inference knowledge learned by the chosen ML algorithms. However, in one case in particular (section 7.5), a quite detailed strategy has been produced through a knowledge engineering-type approach.

While this is not a wholly satisfactory state of affairs, this 'manual' definition of strategic knowledge would seem to be necessary here to provide a context from within which to evaluate the machine-learned inference knowledge.

6.2 Knowledge-Based Systems and Design Reasoning

The second chapter described the logic underpinning design reasoning; to be successful, a design KBS has to emulate this reasoning in some manner. The synthesis of design solutions, at least for configuration problems, was characterised as relying on abductive reasoning. The implementation of form of reasoning is not well-understood; in particular, three questions were raised in chapter 2 concerning the manner in which successful abductive inferences are made:

- what triggers abductive reasoning?
- what mechanism controls the generation of abductive explanations?
- upon what criteria is the judgement of the 'best' explanation based?

With a design KBS constructed as outlined in the previous section, answers can be suggested to these questions.

6.2.1 Triggering Abduction

The first problem is the question of what circumstance triggers an abductive process. In this case, the arrival of a new design problem instigates a new design process, so this provides the effective trigger. By supplying this specification, and invoking the KBS, the user implies

that there is a *need* to explain this specification in terms of a design solution. Consequently, the reasoning is initiated, and the KBS begins its task of generating a suitable cause of this effect.

This specification is to be described in terms of one of the two representations presented in the previous chapter. The assumption is made that this specification is accurate and complete when supplied at the outset of the process.

6.2.2 Abductive Reasoning Mechanism

Once invoked, the KBS must attempt to infer an appropriate 'cause' of the given specification. In this context, the only thing that will pass muster (and be useful) as a cause is a conceptual design of a complete fluid power circuit. This is described in terms of a circuit template and a set of values, one associated with each solution element, according to the representation described in the previous chapter.

As described above, the reasoning mechanism by which a solution is generated is one of 'probable deduction'. The heuristic knowledge is held to be valid, producing suitable explanations of the specification effect. This heuristic knowledge consists of some (manually supplied) strategy and inference knowledge to implement this design strategy. Hence, this inference knowledge must be some body of knowledge with which, when given a design specification, it is possible to deduce a design solution (through the correct application of the strategic knowledge):

Body of machine-learned inference knowledge
Design specification

Deduce: a design solution

So as to be appropriate for this deduction, and infer a solution from a specification, the inference knowledge must manipulate elements of the domain knowledge. Consequently, this knowledge must be expressed in terms of the attributes, and their values, devised to represent the specifications and solutions. Furthermore, in its representation, the inference knowledge must also express some sort of procedural transformation from one to the other – *if* this specification *then use* this solution – to allow the process to move from specification towards solution.

The actual representation of these heuristics themselves depends on the machine learning algorithm applied: as seen in chapter 4, different algorithms produce knowledge of different forms, some of which would seem more appropriate for expressing inference heuristics than

would others. In the next chapter, the use of artificial neural networks to capture this knowledge will be described – in this case, then, the knowledge is represented by a trained network. Chapter 8 discusses the use of a classification construction algorithm, which represents its knowledge in the form of rules. Finally, chapter 9 describes the use of a conceptual clustering algorithm, resulting in knowledge in the form of a concept hierarchy. The advantages and the drawbacks of each of these representations will be outlined in the respective chapters.

6.2.3 Making the 'Best' Decisions

The 'best explanation' of any particular design specification will be a design solution that will achieve it. A design KBS constructed in the manner outlined above will only be successful in producing best explanations if the knowledge that it contains accurately reflects and describes the conceptual design task in hand.

The domain knowledge must capture and express the essentials of the problem. The specification representation must contain all of the information that is necessary for producing the intended solution. Not only must the attributes be appropriate to the task, but so too must the range of values defined for each attribute. Hence, the specification and solution representations, taken together, ought to describe a 'self-contained' problem - 'external' information should not be needed in order to select the 'best' solution.

As seen in the previous chapter, much effort has been expended to try to meet these criteria for suitable representations. However, as was also seen, this is not an easy task; for the specifications in particular, many different representations, expressing quite different information at different levels of detail and abstraction, could be proposed and would appear to be valid. The problem of domain knowledge representation has been eased somewhat by the restriction of consideration to circuits based on a single template, and by concentrating upon expressing functionality, rather than any other aspects of the task, in the specifications and solutions. Nonetheless, the only manner by which the representations might be 'proved' to be appropriate would seem to be through their use in a successful design KBS.

(It is assumed that, at the start of the design process, the user of the KBS will supply a complete and accurate specification in terms of the attribute-value pairs prescribed by the representation. In so doing, the user is providing the initial working knowledge, expressed in the expected manner.)

The heuristic knowledge, both inference and strategic, must encapsulate the manner in which good solutions are derived using the information in the specification. As mentioned

above, the strategic knowledge is, on the whole, simple in nature, and suggested by the form of the knowledge learned by the available ML algorithms. Even so, if this knowledge is poor, then good solutions will not be found.

As discussed in section 4.7, there are a number of factors which influence the quality of any knowledge gained through machine learning. Since this is to be the source of the inference knowledge in this case, these factors will now be discussed with reference to this particular problem.

Data Representation

The learning problem must be presented in an appropriate form to the algorithm; typically, algorithms expect consistent sets of attribute-value pairs. The representations of the problem were developed with this in mind, and so would seem to fulfil this criterion. In addition, these representations must be used to describe the archive examples in a consistent manner so as to make evident the regularities and associations within the examples. Some of the issues surrounding the construction of the archive, mentioned in section 5.3, arise due to this need for the consistent use of the representations.

Data Quantity

The amount of examples must provide some degree of coverage across the full range of associations between specification 'effects' and solution 'causes' that exist in the domain, if the algorithm is to be able to learn generalised knowledge. Without this, although the learned knowledge might be able to make good decisions in cases that are similar in nature to those in the archive, a poor response would be made to less familiar specifications. To be considered as embodying expertise, there would seem to be a need for some degree of 'completeness' in the response of the system. It would not be enough to make an 'expert' response to a handful of individual cases; instead, good explanations would have to be made in response to a large proportion of the potential specifications.

It is also a requirement that the representations used must be at an appropriate level of description that will allow such generalisations to be recognised and learned.

Data Quality

Noise within the examples can hamper successful learning; as explained in section 5.3, a great deal of effort has been expended to try to ensure that the examples are correct and consistent with one another.

A related issue concerning the quality of the examples arises in this case due to the fact that they do not originate from a common source. The idea of the 'best' (or at least, a good) choice of elements will hopefully be embodied within the machine-learned heuristics. For this to be so, the archive must exemplify this sort of 'best' decision-making. Hence, it is assumed that the examples in the archive have been produced by design experts of a similar proficiency, and all are of *successful* design episodes. However, this is unlikely that the archive displays this sort of consistency in the quality of examples: some designs will doubtless represent better solutions to their specifications than will others.

Representation of the Learned Knowledge

As discussed above, the heuristics must be represented in a manner that is appropriate to this design task. Different ML algorithms represent their learned knowledge in different forms. Here, the inference heuristics as a whole must express and permit some sort of transformation from specification to solution. In subsequent chapters, the use of ANNs, rules and a concept hierarchy to represent heuristic design synthesis knowledge will be described. In the context of appropriate strategies, each of these forms can be seen to effect the required transformation. Some indication of whether or not they provide appropriate representations of this type of knowledge can be gained from the behaviour of the KBSs which use them.

Learning Bias

A particular inductive ML algorithm is tailored to search for generalisations of a certain sort. Accordingly, the algorithm may or may not produce knowledge that is appropriate for performing this particular task, even though its representation may appear suitable for the task. Again, some indication of this is provided by the operation of the developed KBSs. Chapter 9 describes the use of an algorithm which learns knowledge that is ostensibly useful, but which, after consideration of the behaviour of the algorithm, turns out to be wholly inappropriate.

These, then, are the factors that provide a basis for the hypothesis that ML algorithms are able to learn abductive heuristics for the task of designing fluid power circuits. However, they do not guarantee that it is possible. Success depends on, amongst other things, whether these representations are useful ones for describing design heuristics in this domain, the quality of the examples, the quantity of the examples, whether the learning task as it is defined is plausible, and whether the ML algorithm used is appropriate. Indeed, it is the purpose of this experimentation to try to decide whether it is possible, using current ML techniques, to learn these heuristics.

6.3 The Construction of a Design KBS Using ML

The preceding sections have outlined the basis for constructing a design KBS using inductive ML for acquiring the necessary inference heuristics. Section 4.8 listed the sequential stages involved in any general application of inductive ML; this can now be re-stated in detail for this particular learning task, and placed in the context of building a design KBS.

The domain knowledge for the conceptual design of fluid power circuits has been established, and the design examples in the archive have been described accordingly. The subsequent steps in building the KBS are as follows:

1. Manually generate a design strategy that, given the appropriate inference knowledge, will produce solutions from specifications (as mentioned above, this strategy may well be influenced by the desire to investigate the use of a particular ML algorithm, and the type of knowledge that it learns). The necessary inference knowledge should be easily identifiable from this strategy.

2. Propose a particular inductive ML algorithm as suitable for learning each element of the inference knowledge. (Although different algorithms could be used within the context of the same strategy, giving different representations of inference knowledge within the same KBS, in the work reported here, a single type of algorithm has been applied during the construction of each KBS.) For each of these learning tasks, the relevant information from the archive examples is identified and extracted, and, where necessary, re-described to comply with the format expected by the chosen algorithm, in order to form the training data for the task.

3. For each learning task, appropriate training parameters are selected and the algorithm is applied to the training data, until satisfactory results are obtained.

4. The results of these learning processes can then be incorporated with a computer system that implements and controls the strategy and provides a user interface, giving a KBS for this task.

Once the KBS has been built it can then be tested, hopefully allowing some general conclusions to be drawn about the value of this approach. Since, as has been explained, there are a number of what might be termed 'working hypotheses' in this process (the specification and solution representations, the strategic knowledge, the ML training parameters, etc.), it should not be concluded from the poor performance of one particular

KBS that this ML approach is unsuited to the task of constructing KBSs. Rather, in such circumstances, it is necessary to consider the behaviour of the KBS, and the form and content of its knowledge, and to try to determine whether, under favourable conditions, the approach *could* result in successful design KBSs.

In step 3 above, the task of judging when the algorithm has satisfactorily learned the required knowledge presents something of a problem. For inductive ML, the usual method of doing this is to partition the available examples into training and testing data sets. If, following a period of training, the learned knowledge suggests appropriate responses to all (or, at least, a sufficiently high proportion) of the testing data, then the knowledge is considered to be satisfactory and the learning process a success.

This approach presents a number of difficulties in this and similar design contexts, though. First, with few examples available, setting aside a portion for testing purposes seems to be rather extravagant.

Secondly, given a particular design specification, there may be two or more equally valid design solutions. Hence, perfectly good solutions produced by the application of the knowledge might not correspond to solutions suggested in the testing set, and, as a consequence, the knowledge appears poor. (This difficulty, though, would seem to be indicative of more serious, theoretical problems with the use of ML to learn this sort of knowledge, which will be discussed later in chapter 10.)

In response to these difficulties, the method adopted was one of inspecting the behaviour of the learned knowledge across the *training* set, and when the algorithm was considered to have learned this data satisfactorily (i.e., the knowledge produced the expected response to all inputs in the data), then the training was considered complete. It is recognised that this is a rather serious shortcoming in the methodology – no measure of the degree to which the algorithm is generalising across the range of problems is used to control the learning process.

In addition, it would be necessary to perform some testing of the developed KBSs. Given the above problems with testing the during the learning of the inference knowledge, the method adopted for testing of the complete systems involves making a manual appraisal of the design solution suggested by the systems. This can be time-consuming, since many examples must be considered to acquire some accurate reflection of overall performance, and it requires someone with the necessary expertise to be able to judge the quality of the designs. However, in the light of the difficulties of conventional testing, it became apparent

after the implementation of the first KBS (section 7.3) that this approach was that needed here. Accordingly, a testing method of this type was devised.

6.3.1 A KBS Testing Method

This method, then, is based on human appraisal of system results; that is, a human with the necessary expertise in the domain assesses the suggested design solutions. The method allows a measure of the *comparative* worth of systems to be gained – hence, it is assumed that the systems are directly comparable. In other words, the systems accept specifications described in the same manner and produce design solutions at the same level of representation, and also that their inference knowledge components have been learned using the same data.[7]

A set of 20 test cases has been devised. A test case consists of a specification described according to the static state representation: hence, each case is described by values of 14 specification attributes. No test case shares a specification with an example in the archive. Each of the test cases is presented in turn to the system, and its output solution recorded.

Appraising Solutions

Rather than simply deciding that a particular response is wrong, interest lies in the degree to which it fails to meet the specification. A system that generates a high proportion of 'nearly right' solutions is more interesting and worthy of further investigation than one that consistently generates utterly wrong solutions. To try to capture these subtleties in system response, each solution is manually appraised and characterised according to the following criteria:

1. Solution is 100% correct.
2. Solution over-engineered, but it 'does the job'. In other words, it contains elements that duplicate the required functionality; the solution performs as desired, but contains too many elements.
3. The selected solution elements are correct, but the solution is incomplete. Certain desired functionality is not provided by the solution.

---

[7] Since, as will be seen, the development of the first KBS led to a change in the representation of specifications that was used, and subsequent systems were developed using more archive examples, this system was not thought to be directly comparable to the systems developed subsequently. As a consequence, this testing method was not applied to this system.

4. A correct and complete circuit is a subset of an incorrect solution. The solution provides all the functionality demanded by the specification – but also functionality beyond that requested.
5. Some of the selected solution elements are correct, some are incorrect and some necessary elements are missing. Some of the requested functionality is provided and some is missing - and some unwanted functionality is also provided.
6. Solution is 100% incorrect. None of the desired functionality is provided.

In general, these interpretations are ordered – the lower the category number, the better the outcome. However, it is not always immediately apparent which of two solutions lying in adjacent categories is the better (particularly with solutions lying in categories 3 and 4). This decision seems to depend on the appraiser's judgement of the difficulty of the task 'remaining' in order to transform the given solution into a category 1 solution. Further blurring of the ordering occurs as the categories fail to take into account the *degree* to which a solution in a particular category is unsatisfactory. For example, is a category 3 solution lacking 3 elements to make it correct necessarily better than a category 4 solution having but a single incorrect element? These problems would seem to be the symptoms of a deeper malaise, the fact that beyond a completely correct solution, which functions as per the given specification, any attempt to assign degrees of 'goodness' to partial or incomplete solutions is bound to be highly subjective. These caveats should be borne in mind and the testing results given for the systems in following chapters should be viewed with a certain amount of caution - hopefully, however, they should give some impression of the relative merits of the systems.

6.4 Summary

This chapter has presented the sources of the various types of knowledge required for a complete KBS devoted to the design of fluid power circuits. In this case, the inference knowledge is to be generated by the application of machine learning algorithms to the archive of example designs. The other categories of knowledge are derived from 'manual' sources.

Also discussed in this chapter has been the manner in which a KBS constructed in this way can be seen to implement the type of reasoning required for design synthesis. The system potentially offers a mechanism by which abductive reasoning is invoked, and a 'best' solution cause of the design specification is suggested. The success of this reasoning is dependent on the quality of the knowledge within the system.

Finally, the methods by which such a KBS is constructed and then tested have been presented.

Following on from this, the next chapter describes three KBSs, each developed using ANNs to capture inference knowledge, Chapter 8 describes a system built using a classification construction algorithm, and Chapter 9 a system developed using a conceptual clustering algorithm. The results of testing each of these systems, along with considerations of the nature and content of the knowledge they contain, allow a general discussion of the value of the inductive ML approach to the capture of design heuristics, as found in chapter 10.

7 The Capture of Design Heuristics using Artificial Neural Networks

This chapter describes three KBSs that have been developed in which the necessary inference knowledge has been acquired through the use of ANNs. Each of these KBSs is described in terms of the form and content of the domain, inference, strategic and working knowledge that constitutes the system. The actual implementation of the system is also described, with particular reference to the application of the ANNs, along with some results generated by the system and a general discussion of its behaviour.

First, however, the rationale that suggests that ANNs may be appropriate for the capture and expression of inference knowledge will be discussed, as will the method of applying ANNs, since, in this context, their use presents particular difficulties not associated with the other ML algorithms.

7.1 ANNs and Inference Heuristics

Feed-forward artificial neural networks are considered to be potentially useful for learning and expressing design heuristics. In particular, they have a number of qualities which suggest that they might be suitable for this task:

- ANNs are able to express associations between a set of input variables and a set of output variables. As such, this corresponds to the sort of association that might hold between specification attributes and solution elements, with simultaneous consideration of a number of the former being necessary to select from amongst the latter. In this manner, they represent a quite sophisticated method of expressing the contextual information which would seem to inform a particular heuristic decision.
- Learning mechanisms, such as the Backpropagation algorithm, exist for training a network to learn generalised forms of these associations from a body of example data.
- ANNs form concise representations of the learned knowledge: once trained, the operation of ANNs is fast. In addition, they have been found to cope adequately with data that contains noise (to a certain extent).

However, the application of ANNs to learning tasks can be particularly difficult and time-consuming, perhaps more so than is the case for the other classes of ML algorithm. When they fail to train satisfactorily, this may be the result of a number of factors: a poor choice of ANN topology or parameters, inadequate training data, or even the definition of a learning task in which there is no deterministic association between inputs and outputs. The virtually incomprehensible nature of a trained ANN - a collection of computational units and weighted links between them - offers few clues as to the reasons for the failure to learn. Often, the only approach is to alter some of the learning 'variables' and repeat the training, until either the ANN trains successfully or persistent failure causes the attempt to be abandoned.

7.2 The Application of ANNs

As described in section 6.3, given a particular design strategy, the first task is to identify the inference knowledge, and, hence, the machine learning tasks, required for its implementation. Examples of this inference knowledge must then be extracted from the archive to form the basis of the training data. If necessary, these examples must be re-described to comply with the format expected by the chosen algorithm. This re-description must be done with care if the learning task is to be faithfully communicated without the loss or addition of information, thus altering the nature of the learning task presented to the algorithm. This could render the task unlearnable, or result in the learning of knowledge other than that required to implement the strategy.

In the case of ANNs, if they are not already expressed in this form, the input and output representations must be encoded appropriately using quantitative values: any symbolic values have to be rendered numerically. The sizes of these *encodings*, that is, the number of terms required to describe the learning task to the algorithm, define the dimensions of the input and output layers of the network.

The next step is to define the remainder of the ANN topology (that is, the numbers of additional network layers and the numbers of units in each) and values for the training parameters (learning rate, etc). There is little assistance available for the selection of these. Most attempts to use ANNs involve a 'trial and error' approach — the correct topology and parameters have been selected when the trained ANN performs satisfactorily. However, as discussed above, determining whether or not satisfactory performance has been attained in this particular context is not a straightforward task.

Furthermore, before testing can be performed the ANN response has to be interpreted. Each output from an ANN is a real number (usually lying between 0 and 1). If the outputs correspond to symbolic elements, then this number must be interpreted. Here, the inference task involves selecting qualitative solution elements, so some interpretation is going to be necessary. A value of, say, 0.64 may be taken to be some indication of the likelihood of the corresponding element being present in a solution. However, at some stage during the design process, unequivocal decisions have to be made, since solutions have to consist of definite elements.

One approach is to apply a hard *threshold*, asserting, for instance, that elements assigned values of 0.5 and above are definitely included in the solution, with all others excluded. Alternatively, some combination of thresholds and likelihoods might be used in order to generate a number of candidate solutions.

As will now be seen, ANNs have been applied in the construction of three KBSs, each employing a quite distinct strategy for performing this design task. In each case, a number of ANNs were trained, using different topologies and parameters and those felt to give the best performance upon their training data were used to provide the required inference knowledge.

7.3 KBS 1 – *Explicit Functional Reasoning*

This strategy adopted for this first KBS involves the use of explicit functional reasoning about the design solutions in this domain – the knowledge model of the design system in which this reasoning is applied is as follows.

7.3.1 Strategic Knowledge

The strategic knowledge for each system, then, is the high-level methodology that is applied in order to translate the design specification into a suitable solution. For this KBS, this design strategy is shown in Figure 11. In this, the template is assumed to form the basis of all solutions. Given the specification, the first task is decide what additional functionality (that is, beyond that supplied by the template) the solution must provide. That done, the next task is to select those solution elements that will provide this functionality when placed in their prescribed positions in the template. The final, trivial task (not shown) is that of inserting the chosen elements into the template.

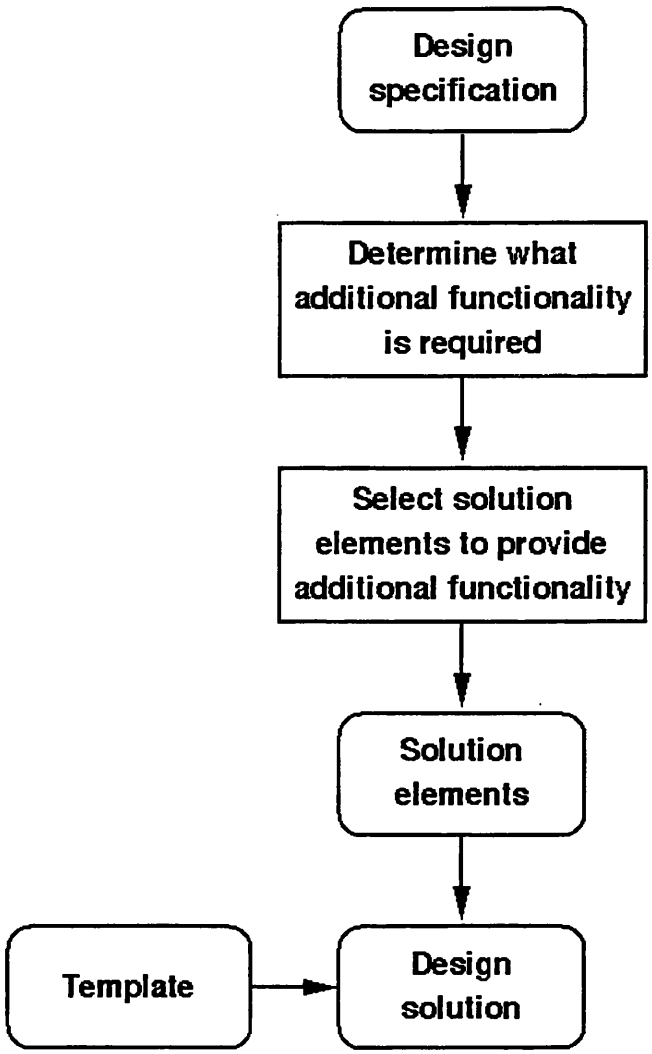


Figure 11. Functional reasoning design strategy (KBS 1).

7.3.2 Inference Knowledge

The inference knowledge in this strategy is as follows:

- the knowledge for determining the additional functionality required for a solution. This takes the form of a translation of the specification into a set of functions.
- the knowledge for choosing an appropriate solution element to fulfil the functionality. For each function, there is an associated, independent choice of element, made on the basis of the information contained within the specification.

7.3.3 Domain Knowledge

The domain knowledge in this model consists of knowledge of potential design specifications, described according to the temporal state representation, and of potential solutions, in terms of the template and solution elements. In addition, for this model, domain knowledge is needed in the form of the additional functionality that circuits may implement. This has resulted in the introduction of 9 terms to describe this functionality, as follows:

- *control the speed of actuator extension.*
- *control the speed of actuator retraction.*
- *control the speed of both actuator extension and retraction.*
- *prevent excess pressures occurring within the system.*
- *hold the load stationary at any user-specified position during the functioning of the system.*
- *hold the load stationary in the event of system failure.*
- *prevent cavitation occurring in the system.*
- *cater for peak pressure demands occurring in the system.*
- *prevent the load overrunning in its motion.*

As mentioned in the previous chapter, these terms were handcrafted from a familiarity with the domain; it should be noted that they are domain specific, referring to concepts expressly related to fluid power systems.

Each function in this list must be explicitly associated with those solution elements that may be used to fulfil it – this is further domain knowledge that is necessary for this approach to this design task.

7.3.4 Working Knowledge

The working knowledge in the model is in the form of the current design specification, and the selected functionality and solution elements.

7.3.5 Implementation – Functional Reasoning KBS

Feed-forward ANNs would be applied to learning the inference knowledge required in this case. Once trained, the networks would be embedded within suitable code for controlling the strategy and manipulating the working knowledge. The particular network training algorithm

selected was that of the *Time Delay Neural Network* (Waibel *et al.*, 1989). This algorithm uses a modification of the Backpropagation algorithm to enable it to learn the temporal relationships that exist amongst attributes. In effect, it is able to accept as input data values that describe the same attributes at a number of sequential points in time, and relate these to a time-invariant pattern of outputs. In this way, the algorithm was thought to be suited to learning about the relationships between the dynamic specifications, described, using the temporal state representation, over a number of sequential states, and the static functions and solution elements.

This initial implementation would consist of the domain function selection network, and solution element selection networks corresponding to the functions *prevent excess pressures occurring within the system*, *hold the load stationary in the event of system failure*, and *prevent the load overrunning in its motion*. These functions were considered to be the best exemplified by the data, and would serve to investigate the viability of the approach. The other functions, if required, would be satisfied by the selection of an appropriate default solution element from the subset associated with each function. Figure 12 depicts this implementation model.

Training Data

The training data were produced from a total of 16 archive examples, with each example described according to the appropriate specification and solution representations. Additional information is added to each example in the form of the domain functions that the solution is considered to provide, and the solution element in the solution that provides each function.

Since ANNs can deal only with numerical data (usually real values between 0 and 1), before training can commence the archive examples must be re-expressed in such terms. To transform the specifications, consideration must be made of the type of value used to express each attribute. If the attribute can assume discrete, binary values (for example, *yes* and *no*, or *high* and *low*) then it can be coded as a single input unit, with a value of 1 corresponding to, say, *yes* and 0 corresponding to *no*. (An alternative encoding might be to use 1 and -1, or 0.5 and -0.5, each placing a slightly different emphasis on the data.)

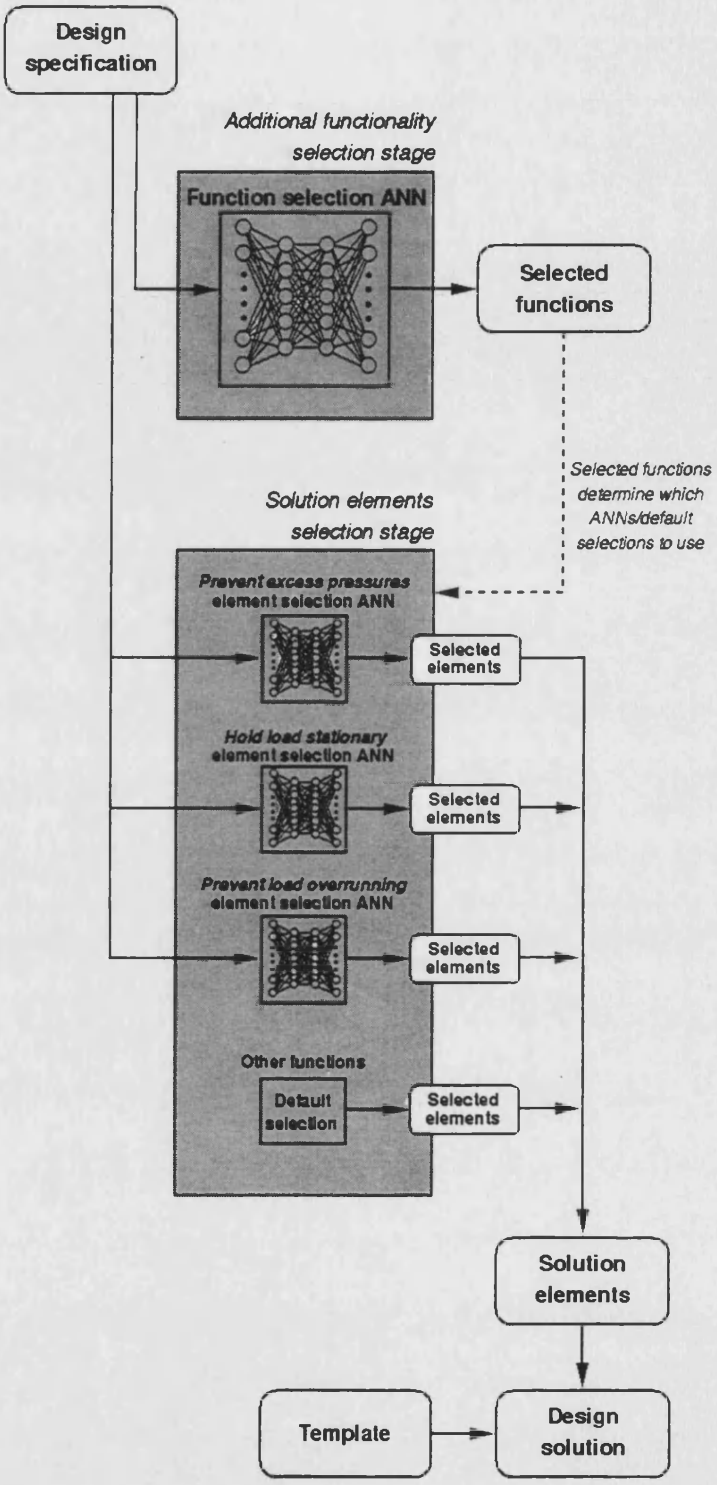


Figure 12. Implementation model for functional reasoning KBS (KBS 1).

The encoding of real-valued attributes presents a greater number of alternative representations. The value can be normalised between 0 and 1, and presented directly as input. However, it may be the case that the range of the parameter can usefully be divided into symbolic sub-ranges. For example, for an attribute representing *load* magnitude, the

range of values might be divided into 'fuzzy' *low*, *medium* and *high* categories (as in the static state specification representation), if it is felt that these provide the necessary load magnitude information for performing this design task. If so, it may be better to translate the values into this form, since doing so can improve the generalisation capabilities of the network by making regularities in the data more explicit. To do this, however, these 'useful' divisions must be known (and so, introduces further domain knowledge into the system).

Then, though, there remains the problem of representing discrete, but non-binary, values. Once again, a single node might be used, with, say, 0 representing *low*, 1 representing *high* and 0.5 representing *medium*. This encoding would seem appropriate in this case, as the load attribute value has been transformed into one of a set of ordinal discrete values, and this natural ordering is reflected in the choice of input value. Alternatively, 3 separate input units could be used, each corresponding to one of the possible values (which may be a more appropriate choice if the set is not ordered).

For a more detailed discussion of the techniques of data 'pre-processing' of this sort, with more examples of the choices that may have to be made and suggested encodings of data, see (Bishop, 1995). This reference also includes a discussion of techniques for eliminating from the data elements that provide little information. This reduces the dimensionality of the data, which, in turn, can improve generalisation during learning, and, although not applied in the current work, is a common strategy when using machine learning approaches.

In this case, then, to encode the specifications, for every time period, each of the real values of the *load* magnitude, *speed*, *distance* and *duration* attributes is represented by membership of one of three fuzzy sets (nominally *low*, *medium* and *high*). The value of the *direction* attribute would be represented by a binary digit, 1 indicating load extension, and 0 load retraction. Similarly with the value of the *plane* attribute (1 indicating non-horizontal motion, 0 horizontal), and the value of the *velocity control* attribute (1 indicating control is required, 0 that it is not). In total, then, fifteen values are used to describe each state of a specification.

The additional functionality of each example is represented by a set of nine binary values, one for each of the basic functions, a value of 1 representing the presence of that function in the circuit, 0 its absence. Each of the solution elements is also represented by a binary digit (1 presence, 0 absence) for every example.

To illustrate this, the encoding of the archive example discussed in chapter 5 will be considered. The circuit is shown in Figure 13, and the specification, according to the

temporal state representation, is shown in Table 7. The specification is encoded into the pattern shown in Table 8. Note that values have to be supplied for time and duration – these are given *medium* values, in the absence of other information.

attribute	***state 1 values***	***state 2 values***
load [mass × 9.8]	8722	8722
distance	0.61	0.61
speed	-	-
duration	-	-
direction	extension	retraction
plane	non-horizontal	non-horizontal
velocity control	no	no

Table 7. Example specification.

attribute		***state 1 values***	***state 2 values***
load	*low*	1	1
	medium	0	0
	high	0	0
distance	*low*	1	1
	medium	0	0
	high	0	0
speed	*low*	0	0
	medium	1	1
	high	0	0
duration	*low*	0	0
	medium	1	1
	high	0	0
direction		1	0
plane		1	1
velocity control		0	0

Table 8. Example specification encoded for ANNs for KBS 1.

In this particular solution, 3 of the domain functions are considered to be satisfied:

- *prevent excess pressures occurring within the system*
- *hold the load stationary at any user-specified position during the functioning of the system*
- *prevent cavitation occurring in the system*

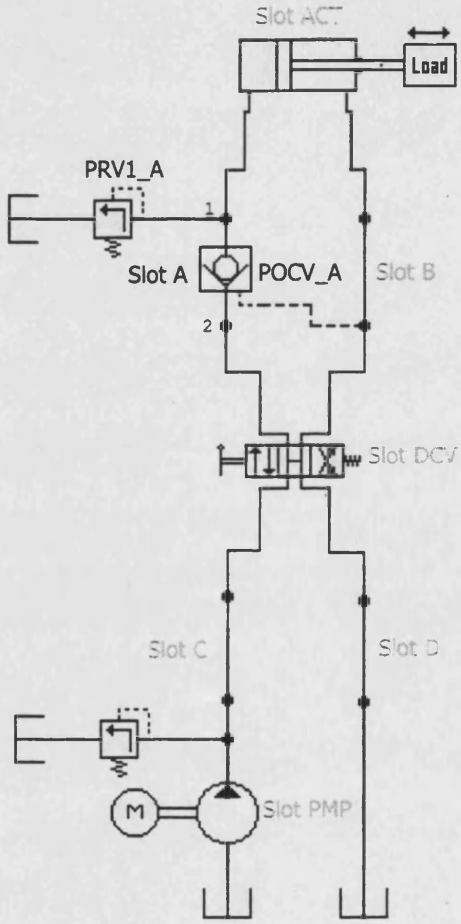


Figure 13. Example fluid power circuit.

So, the encoding of the domain functions in this case is as shown in Table 9.

Next, each of these functions needs to be associated with some element of the solution:

- *prevent excess pressures occurring within the system* is considered to be satisfied by the element *PRV1_A*.
- *hold the load stationary at any user-specified position during the functioning of the system* is considered to be achieved by the use of element *POCV_A*.

- *prevent cavitation occurring in the system* is also considered to be satisfied by the use of *PRV1_A*.

Finally, the solution elements need to be encoded (Table 10).

basic function	*value*
control speed of extension	0
control speed of retraction	0
control speed of both extension & retraction	0
prevent excess pressures occurring	1
hold load stationary	1
hold load on failure	0
prevent cavitation	1
cater for peak demands	0
prevent load overrunning	0

Table 9. Example encoding of domain functions.

solution element	*value*
PRV1_A	1
PRV1_A&B	0
POCV_A	1
CBV1_A	0
CBV1_A&B	0
CBV2_A	0
PCMP_C&PROP_DCV	0
DECV_A&B	0
MO_A	0
MO_B	0
CVC_A&B	0
VPCRV_D	0
PRV2_A&B	0
VDP_PMP	0
MOT_ACT	0
PROP_DCV	0
CC_DCV	0

Table 10. The encoding of the example solution.

Again, it should be emphasised that, since the optimal encoding of representations is dependent on the particular problem in question, there is little assistance to be found in the literature for making these decisions beyond suggesting the sorts of encodings that might be considered. As such, these encodings are best viewed as working hypotheses within the system.

These encoded data are then combined into the appropriate training sets for learning the inference knowledge required for the prototype KBS implementation:

- for the *function selection ANN* - a set containing the combination of encoded specification (as input pattern) and domain function (as output pattern) information for each of the 16 archive examples. Each input pattern consists of 15 values for each state in the corresponding example specification, and each output pattern consists of 9 values.
- for each of the 3 *solution element selection ANNs* implemented - a set containing the encoded specifications (as input patterns) and corresponding sets of solution elements that are used to satisfy the domain function in question, indicating their presence or absence as appropriate (output pattern). Since the training sets exemplify only those cases in which the corresponding function is satisfied, each set could contain any number of patterns up to a maximum of 16. As before, an input pattern is described by 15 values for each state in the corresponding specification. The output pattern contains a value corresponding to each solution element that can be used to fulfil that particular function.

So, then, a total of 4 ANNs have been trained. For the first, the function selection algorithm, the following topology was found to give the best performance:

- an input layer of $15 \times 5 = 75$ units (15 is the number of values required to describe each specification state, and 5 is the maximum number of states required to describe the archive examples; hence, when using the ANN, the description of new design problems would be limited to a maximum of 5 states).
- a hidden layer of 24 units.
- an output layer of 9 units, one corresponding to each of the possible functions.

The solution element selection ANNs corresponding to the functions *prevent excess pressures occurring within the system* and *hold the load stationary in the event of system failure* both have the following topology:

- an input layer of 75 units, as before.

- a hidden layer of 24 units.
- an output layer of 3 units, since each of these functions can be satisfied by the use of one of three solution elements.

Finally, the solution element selection ANN for the function *prevent the load overrunning in its motion* has the following topology:

- an input layer of 75 units.
- a hidden layer of 32 units.
- an output layer of 4 units, since this functions can be satisfied by the use of one of four solution elements.

In each case, the number of units in the hidden layer was determined by trial and error, in attempting to find a topology that could represent the relationship.

These trained ANNs, as the inference knowledge, have been incorporated into a simple software shell, which, by controlling the domain information passed to and from networks, and the order of their use, implements the design strategy, and which also provides a simple command-line interface to the system.

7.3.6 Results

By way of an example of the operation of this system, the specification shown in Table 11 was presented as a new design problem (this specification does not appear in any of the archive examples). The problem here is one of designing a fluid power circuit that is able to perform a 3-state horizontal motion. State one involves extending a load of 100000N a distance of 0.7m in 100s. State two, the retraction of the load, now increased to 100500N, back over the same distance in 200s, and the final stage is to once again extend a load of 100000N over 0.7m.

These values are encoded in the appropriate manner, and fed into the KBS. The function selection ANN is the first invoked in the strategy, producing the output values for each function shown in Table 12. A value greater than 0.5 indicates that the associated function must be achieved in any solution; a value below 0.5 indicates the function is not needed. This threshold value of 0.5 is chosen in this case simply as it is the mid-point of the output range.

In this case, the following functions are deemed necessary for a solution (i.e., they have an associated output of 0.5 or above): *control speed of both extension and retraction, hold load stationary in the event of system failure, prevent cavitation occurring in the system* and *prevent the load overrunning in its motion.*

attribute	*state 1 values*	*state 2 values*	*state 3 values*
load	100000.0	100500.0	100000.0
distance	0.7	0.7	0.7
speed	0.007	0.0035	0.007
duration	100.0	200.0	100.0
direction	extension	retraction	extension
plane	horizontal	horizontal	horizontal
velocity control	no	no	no

Table 11. Specification of example problem.

This information is next used to select appropriate solution elements. Table 13 shows the solution elements chosen to provide each function. These are default elements, with the exception of MO_A and MO_B, which have been selected by the ANN associated with the function *prevent the load overrunning in its motion.*

function selection ANN		
function	*output*	*selected?*
control speed of actuator extension	0.089603	×
control speed of actuator retraction	0.304268	×
control speed of both extension and retraction	1.000000	✓
prevent excess pressures occurring in system	0.001198	×
hold load stationary at any user-controlled position	0.000001	×
hold load stationary in the event of system failure	0.969457	✓
prevent cavitation occurring in the system	0.999903	✓
cater for peak pressures occurring in the system	0.000000	×
prevent the load overrunning in its motion	0.999931	✓

Table 12. Function selection ANN response.

solution element selection	
function	***chosen solution element***
control speed of both extension and retraction	VPCRV_D
hold load stationary in the event of system failure	POCV_A
prevent cavitation occurring in the system	CVC_A&B
prevent the load overrunning in its motion	MO_A *and* MO_B

Table 13. Solution element selection.

In addition to these meter-out groups in slot A and slot B to prevent the load from overrunning, this set of results suggests that to control the speed of both extension and retraction, a variable restrictor valve is required in slot D; a pilot-operated check valve in slot A is needed to hold the load on the event of failure; and a check valve combination across slots A and B would prevent cavitation occurring. Figure 14 shows the resulting circuit solution.

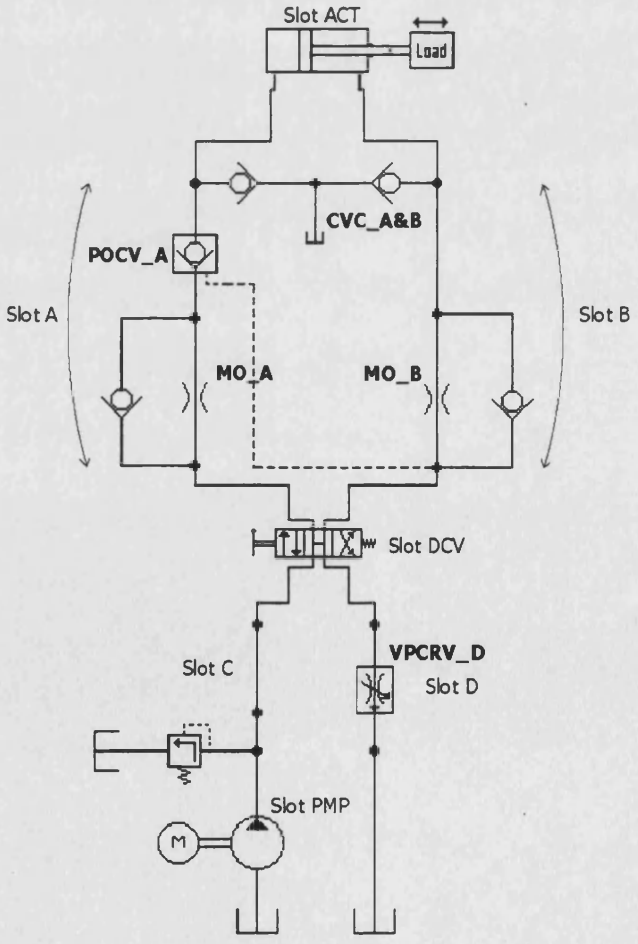


Figure 14. Example solution circuit.

7.3.7 Discussion

It was felt that there were a number of major problems with this approach. A major difficulty lies in the acquisition of the additional domain knowledge that is required to implement this model. The machine learning of the inference knowledge constitutes a second problem area.

Explicit functional reasoning of this sort requires a set of domain functions and so, introduces an additional knowledge acquisition task – these functions must be extracted from an expert or else gained through greater knowledge of the domain on the part of the system developer. In contrast to the specifications and solution elements, which have at least some connection to the external world, these functions would appear to be purely 'internal' constructs of the designer, and so, presumably, are more difficult to extract. In addition, further domain knowledge must be acquired in the form of the sets of solution elements that may be used to achieve each of these functions.

Once this extra domain knowledge has been successfully acquired, it is necessary to interpret each example solution in the archive, determining which of the domain functions it achieves, and the manner in which it achieves them, so as to produce the training data. As a whole, this sort of knowledge would seem to be more detailed and task-specific than could be handcrafted by a system developer familiar with the domain – in other words, it would seem necessary to extract the knowledge from a design expert. With the difficulties surrounding knowledge engineering, this is not a trivial problem and weighs heavily against this sort of design strategy. As a consequence, the strategies used in subsequent systems do not rely upon explicit functional reasoning.

Descriptions of typical ANN applications talk of training data sets that are larger by several orders of magnitude than are used here, and the problem is exacerbated by the (presumed) complexity and high dimensionality (in terms of the number of input and output units) of this particular learning task. The lack of examples of this design process has already been noted, and it is a problem which afflicts all the approaches using inductive ML algorithms described in this thesis.

This problem is not eased by the use of the temporal state specification representation, according to which, the input can increase to an arbitrary size (though, in the admittedly limited range of the archive, it was found that a maximum of 5 states was sufficient to describe all the examples). Furthermore, since the solution element selection ANNs can use only those examples in which the corresponding function is satisfied, even fewer example data are available for their training.

It was felt that the ANNs were too poorly trained to substantiate the approach, so no methodical testing was attempted. Problems encountered with the complexity and size of the temporal state specification representation in the course of this experiment led to its abandonment – however accurate a model for expressing specifications it might be, it seems altogether unsuited for use with ML algorithms. Subsequent experiments would use the static state representation, a less precise, but more compact representation.

7.4 KBS 2 – *Single Stage Transformation*

As a consequence of the lessons learned from the development of the first KBS, a specification representation perhaps more amenable to the approach (the static state representation) was devised, and a much simpler design strategy adopted.

7.4.1 Strategic Knowledge

The design strategy here is very straightforward. Assuming the template, the task is that of making a direct translation of the given specification into a set of solution elements (Figure 15).

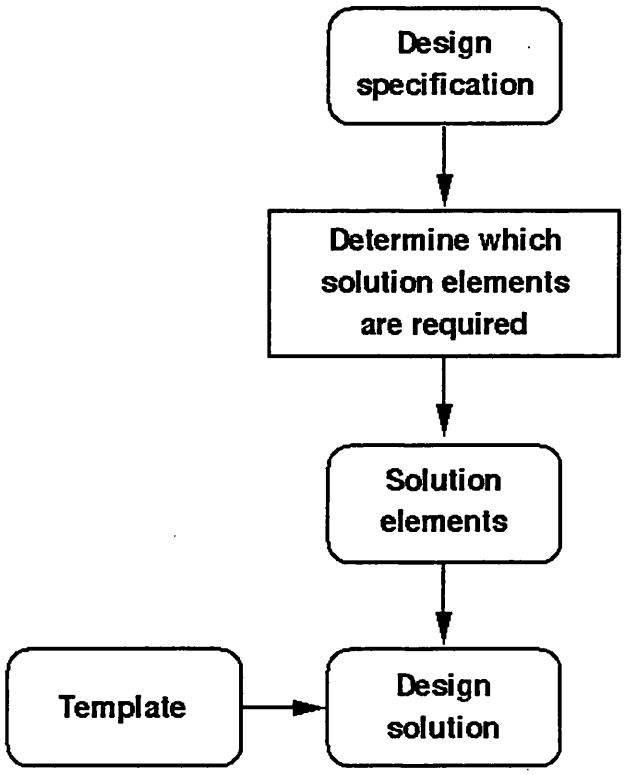


Figure 15. Single stage transformation strategy (KBS 2).

7.4.2 Inference Knowledge

The inference knowledge in this model resides in knowing how to translate in an appropriate manner the specification into the solution elements.

7.4.3 Domain Knowledge

The domain knowledge consists of the attributes of the design specification, and the values that each can assume (static state representation), and the set of solution elements with which to construct designs.

7.4.4 Working Knowledge

The working knowledge in this model consists of the current design specification, and the chosen solution elements.

7.4.5 Implementation – Single Stage Transformation KBS

An ANN, trained using the Backpropagation method, would perform the task of selecting the appropriate solution elements (Figure 16). This would then be embedded within appropriate code for controlling the strategy and the working knowledge. The specific implementation of Backpropagation used is that contained within the *SNNS* package (Zell *et al.*, 1996).

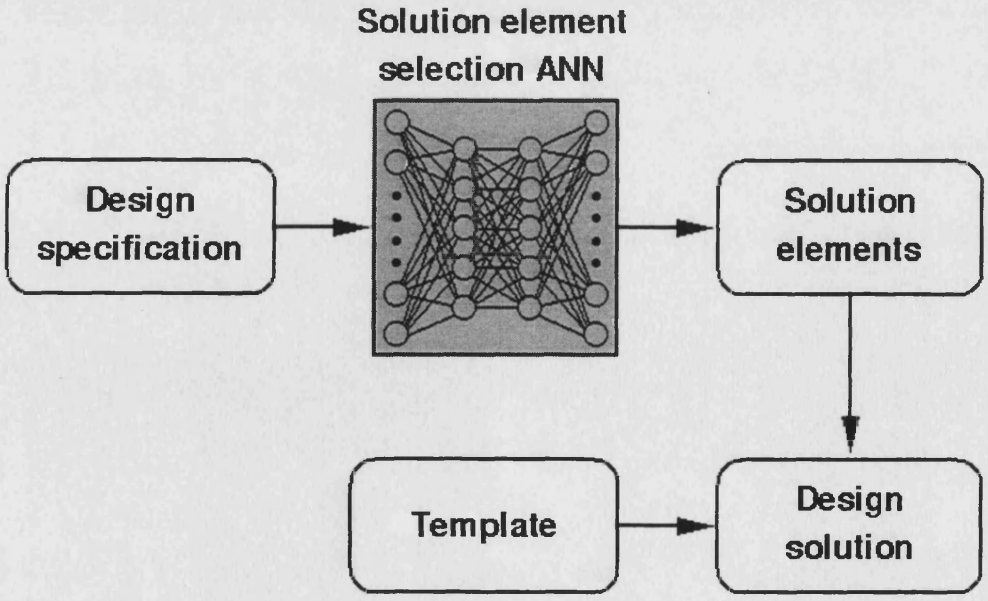


Figure 16. Implementation model of single stage transformation KBS.

Training Data

The training data comprises 30 design examples (the archive had been extended since the implementation of the functional reasoning KBS). As described above, these examples must be encoded in a manner that is appropriate for their use with ANNs. The specification representation in this case, the static state representation, contains two types of attribute, binary-valued and ternary-valued. In this case, binary-valued attributes (for example, *load-independent speed*) were encoded in terms of a single input parameter, having a value of 1 for *yes* and a value of 0 for *no*. The ternary-valued parameters, namely *maximum load/force* and *maximum speed*, were each encoded by a single parameter, having a value of 1, 0.5 or 0, corresponding to *high*, *medium* and *low* values respectively.

The ANN must learn the knowledge that will indicate the presence or absence of each of the solution elements. Accordingly, for the network output, this lends itself to a binary encoding using a separate output unit corresponding to each element, with the value 1 indicating its presence and 0 its absence - and this was the method adopted here. The archive examples could now be re-described to conform to these input and output encodings to produce the training data.

An example of this encoding is shown in Table 14. A design specification, now described according to the static state representation, has the values given in the second column. This is translated into the input encoding shown in the third column. The outputs are encoded as for the previous ANN example (Table 10).

The first step in defining the network itself is simple: since 14 encoded attributes describe the specification, and 17 encode the solution, the network would need 14 input and 17 output units. The next step is to define the remainder of the ANN topology (that is, the numbers of additional network layers and the numbers of units in each) and values for the training parameters. A hidden layer of 10 units was found to give the best performance.

This trained ANN was embedded in control code to implement the strategy. A user interface for supplying a new design specification, accessible from WWW browsers, was written, taking advantage of the ready-made interface facilities that these browsers provide. The browser code invokes the system code, which, in turn, produces results in browser-readable form. Figure 17 shows the specification interface. This interface was re-used in all subsequent systems.

attribute name	*value*	*encoding*
maximum load/force	low	0
maximum speed	medium	0.5
plane of motion	non-horizontal	1
continuously variable speed	no	0
hold load stationary	yes	1
smooth accelerations	no	0
hold load on failure	yes	1
load-independent speed	no	0
control extend speed	no	0
control retract speed	no	0
solution requires motor	no	0
control inertia	yes	1
energy efficiency paramount	no	0
control accuracy	low	0

Table 14. Example specification and its encoding for the ANN of KBS 2.

7.4.6 Results

In general, in a series of trials, the networks were found to be able to learn certain, simple associations from the data. Typically, these learned associations were those between a single specification attribute and a single solution element which were well exemplified in the data. However, network performance was, on the whole, unsatisfactory.

To illustrate the operation of this design system, the response to a test problem produced by one of the trained networks will be described. This test problem has a specification as shown in the second column of Table 15, which is encoded as shown in the third column.

This encoded test specification was presented to the KBS, and its response noted, with ANN output values above a threshold of 0.7 considered to indicate the presence in the solution of the corresponding elements. In this case, the following four elements were deemed necessary: *PCMP_C&PROP_DCV*, *CVC_A&B*, *PRV2_A&B*, *MOT_ACT*.

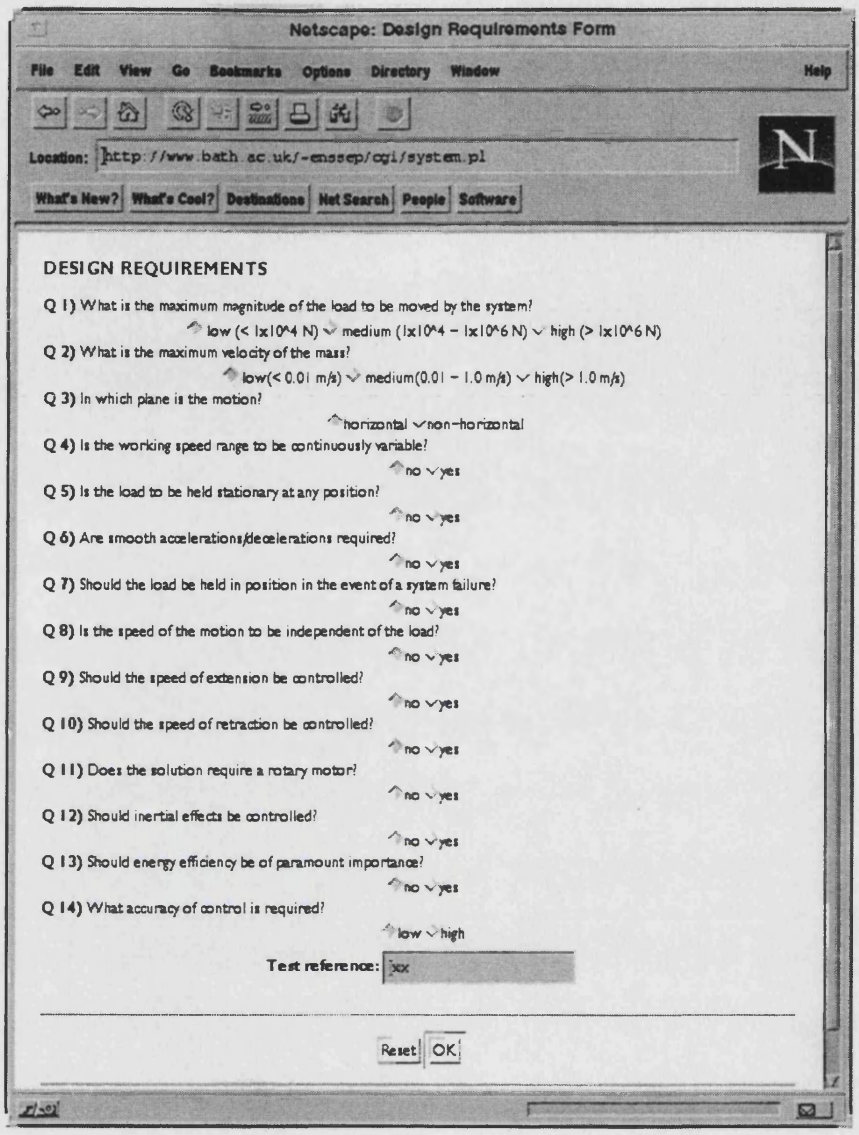


Figure 17. The web-based user interface for the single stage KBS and subsequent systems.

This result was evaluated as follows:

- *PCMP_C&PROP_DCV* – this element can provide continuously variable speed, and so satisfies part of the specification. However, it also provides load-independent speed, not wanted here, and so is a wrong choice of element.
- *CVC_A&B* – this element is used to control the effects of inertia, and so is correct.
- *PRV2_A&B* – similarly, this element is also used to control the effects of inertia, and since it may be used in conjunction with *CVC_A&B* to achieve this, is correct.
- *MOT_ACT* – a motor, explicitly demanded in the specification, correct.

attribute name	*value*	*encoding*
maximum load/force	high	1
maximum speed	low	0
plane of motion	horizontal	0
continuously variable speed	yes	1
hold load stationary	no	0
smooth accelerations	yes	1
hold load on failure	no	0
load-independent speed	no	0
control extend speed	no	0
control retract speed	no	0
solution requires motor	yes	1
control inertia	yes	1
energy efficiency paramount	no	0
control accuracy	high	1

Table 15. Test specification and its encoding.

However, one aspect of the functionality demanded in the specification is not catered for in this solution – there does not seem to be any means of providing smooth accelerations. Overall, then, some of the requested functionality is provided, some is missing, and some unwanted functionality is supplied – this is a category 5 solution, according to the testing method introduced in section 6.3.1.

The overall results of testing this KBS are shown in Figure 18. For this and the subsequent ANN-based KBS (section 7.5), the relative quality of the solutions gained by applying a number of different thresholds to the output has been investigated. If a particular output unit, corresponding to one of the solution elements, has a value above the threshold then that element is considered to form part of the solution. Thresholds ranging from 0.1 to 0.9 in incremental steps of 0.1 were examined; the quality of the solutions to all 20 test cases at each threshold would allow some judgement to be made as to which is the best overall threshold to apply to each system. Hence, in the graph there is a column of results given at each of the examined thresholds. Each column shows how the solutions to the 20 test cases are distributed across the 6 possible categories that are used in the testing method to evaluate

the quality of solutions – in general, the lower the number of the category that a solution falls into, the better the solution is.

Neither this nor the later examination of threshold values is wholly conclusive, beyond the recognition that better results (i.e. with more test cases falling into categories of a lower number) seem to occur when taking the threshold to be somewhere in the range 0.5 – 0.7. This would seem to concur with the general impression when little is known about what the correct threshold should be – too low a threshold, and solutions will tend to include more undesirable elements (so there would be a shift, supported by the evidence towards categories 4 and 5), whereas too high a threshold would tend to preclude necessary elements from solutions (a shift towards categories 3, 5 and 6).

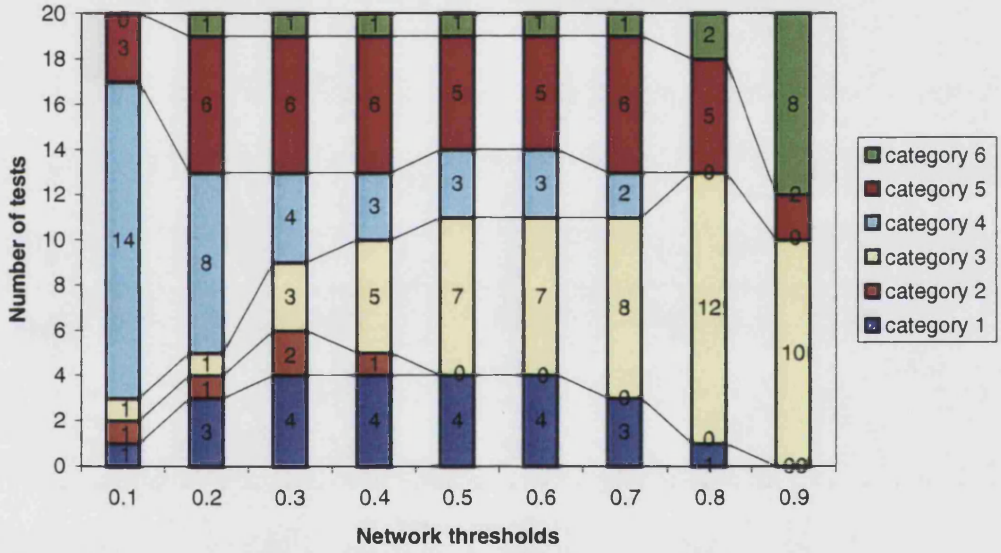


Figure 18. Single stage KBS test results at a range of threshold values.

Considering categories 1 and 2 as being the 'best' results, then, it can be seen that roughly 15 – 20% (across the better thresholds) of the results produced by this system fall into these categories. Focusing instead on the 'worst' solutions, categories 5 and 6, it can be seen that 30 – 35% of the results fall into these categories.

7.4.7 Discussion

The most obvious reason for this less than satisfactory performance would seem to be, once again, the acute lack of data: thirty example cases would generally be considered a number of orders of magnitude too few for a problem involving this number of inputs and outputs. A

consequence of this would seem to be that coverage of the full range of associations is lacking, and so, any ANN trained using this set of data, could not learn full, generalised inference knowledge for the task. Unfortunately, the nature of ANNs and their training mechanisms means that it is not obvious when a network is 'under-trained' in this way. Usually, the application of the network to a set of test data is used to provide some indication of the degree to which the network has been well-trained, but, for reasons stated in the previous chapter, this approach is not particularly practical in this case.

Since an ANN is fully defined (albeit with random connection weightings) from the outset of training, even at this stage it will produce an (almost certainly incorrect) response to an input pattern. The aim of the training algorithm is to alter these weightings so as to represent the associations exemplified in the training data, but nothing in the *structure* of the network itself changes during training to reflect this move from being an untrained to being a trained network. Hence, even when under-trained, an ANN will respond to every input pattern with some output – here, a design solution will be produced in response to every specification. In cases where this response is based on insufficiently modified weightings, it might be preferable if the network were to fail to produce a solution, or to otherwise indicate a lack of confidence in its response.

Put another way, the ANN does not begin from a position of 'knowing' nothing about the problem; rather, it begins from a position of knowing something that is (in all probability) incorrect about the problem. During training, the weightings are modified to produce the correct response to the examples, and so the knowledge of the ANN becomes gradually more correct. However, with few examples, much incorrect knowledge (i.e., incorrect weightings) can remain unmodified at the end of the training phase, and, during subsequent use, this can produce incorrect responses. Hence, the knowledge, such as it is, embodied in an under-trained network consists of some mixture of learned and 'not unlearned' knowledge; it is impossible to distinguish between the application of each type during network performance. This point is reinforced by the fact that in a series of trials with the same training data and the same network topologies, but different random initialisations of connection weightings, trained ANNs were produced that give wildly different responses to the same test patterns. This characteristic of ANN learning would seem to be a very serious limitation of the usefulness of ANNs in problems such as this for which few examples are available and makes it difficult to determine just how much a network is able to learn under these conditions.

However, even if sufficient examples were available, the question remains of whether an ANN (or, for that matter, any available inductive ML algorithm) would be able to learn the

necessary inference knowledge from data of the sort used here. This question will be addressed in greater detail in chapter 10.

7.5 KBS 3 – *Multiple Stage Transformation*

In an attempt to harness the potential of ANNs for capturing design heuristics when few data are available, a new design strategy for the task was introduced. This strategy decomposes the conceptual design process into a series of sequential sub-tasks. The cumulative effect of performing each sub-task correctly should be a complete and correct design solution. Each of these sub-tasks is defined in terms of the selection from amongst a subset of the solution elements in response to the values of a subset of the design specification attributes. These sub-tasks, where appropriate, would be performed by trained ANNs. Since these ANNs handle *subsets* of the problem attributes, each would be smaller than that of the previous experiment - and, in general, the smaller the network, the fewer the data required to train it. However, the immediate drawback of this approach is that the definition of this more detailed strategy requires a greater amount of knowledge of the process.

This decomposition of the design task, then, is intended to be some reflection of how a human designer might approach it, indicating the areas that can be dealt with separately, and the order in which they should be addressed. It also explicitly defines the inference knowledge necessary to implement the strategy.

This strategy has been derived 'manually' by an external human agent, familiar with the domain, but not a design expert – see (Darlington, 1998) for a description of the process by which this strategy was derived. Once more, though, this raises the question of the accuracy of the results of knowledge engineering.

7.5.1 Strategic Knowledge

The design process, then, consists of a number of sequential stages. Given a design specification, the first task is to determine whether or not a rotary motor is required to solve the current problem. The next stage is to select the appropriate solution elements, if any are needed, for providing speed control in the solution. The following stage, selecting some means of controlling inertia in the circuit, itself consists of two stages: the selection of elements for controlling inertia, and then, if these elements produce undesirable side-effects within the circuit, the selection of elements for eliminating these side-effects. The final stage is that of selecting *contextual* solution elements, those that are suggested by the choices made at earlier stages of the process. Each of the decisions made during this process is done

so on the basis of the specification attribute values and, in some cases, the choice of solution elements made at a previous stage. Figure 19 summaries this design strategy and Figure 20 shows it in sufficient detail to allow its implementation.

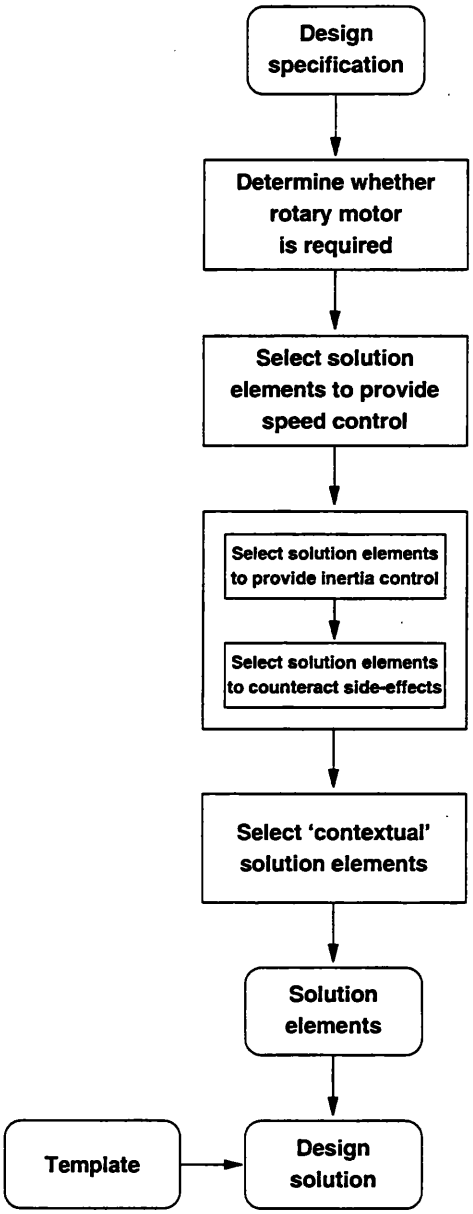


Figure 19. Multiple stage transformation design strategy (KBS 3).

7.5.2 Inference Knowledge

The inference knowledge in this model of the conceptual design of fluid power circuits is as follows:

- the knowledge of whether a motor is required in the solution.

- knowledge of how to select appropriate speed control elements.
- knowledge of how to select appropriate inertia control elements, and of how to eliminate any side-effects that the chosen elements may introduce.
- knowledge of how to select contextual components.

7.5.3 Domain Knowledge

The domain knowledge contains the set of specification attributes, and their possible values (static state representation), and the set of solution elements. In addition, the subsets of the specification attributes and of the solution elements that are relevant to each subtask and knowledge of which elements produce side-effects constitutes domain knowledge.

7.5.4 Working Knowledge

Working knowledge in this model consists of the current design specification, plus the chosen solution elements.

7.5.5 Implementation – Multiple Stage Transformation KBS

There would be trained ANNs implementing the inference knowledge, with the exception of the decision about whether a motor is required in the solution - since the need for a motor is explicitly indicated in the specification, this inference is a trivial one. Relevant sub-sets of the set of specification attributes form the inputs to each network, with the addition of solution elements for determining the side effects elements - Figure 20 contains full details. The outputs from each stage are collated to give the solution.

Once again, the Backpropagation algorithm as part of the SNNS package is used to learn the relationships between inputs and outputs for each of the ANNs.

Training Data

A total of 30 training examples are used to form the data for the networks. Each is encoded as described for the previous experiment. Separate data sets are constructed for each network, containing only the appropriate attribute inputs and outputs, as shown in the detailed strategy (Figure 20).

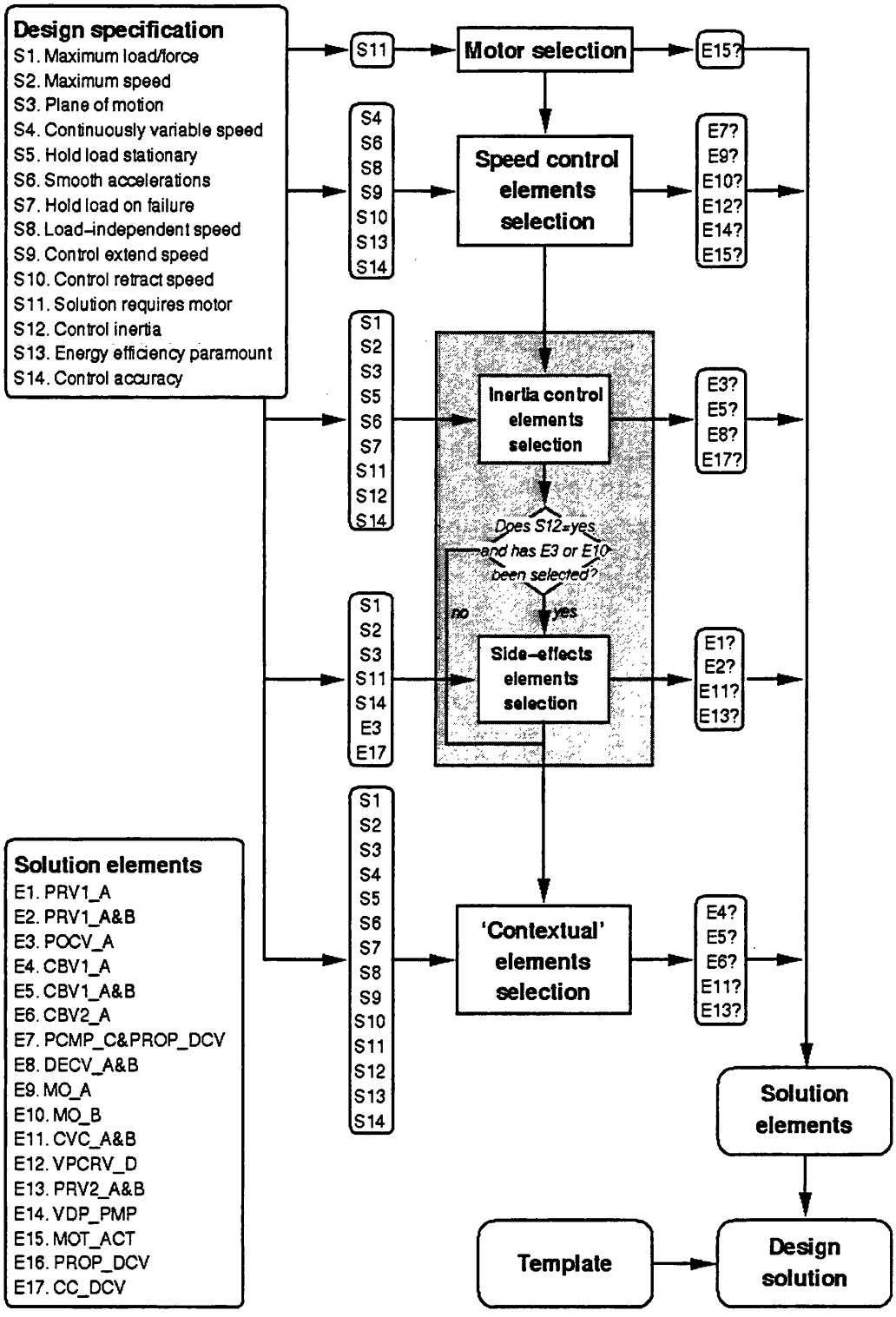


Figure 20. Multiple stage transformation - detailed strategy.

The best ANN topologies found during training are as follows:

- Speed control element selection ANN: 7 input units, 1 hidden layer of 3 units, 6 output units.

- Inertia control element selection ANN: 9 input units, 1 hidden layer of 3 units, 4 output units.
- Inertia control side-effects ANN: 7 input units, 1 hidden layer of 3 units, 4 output units.
- Contextual element selection ANN: 14 input units, 1 hidden layer of 5 units, 5 output units.

7.5.6 Results

To demonstrate the performance of this system, the response to the test specification given previously in Table 15 will be described. Once again, the encoding of this problem is that shown in the third column of this table.

Since the specification indicates that the solution must use a motor, the first task is to add the element *MOT_ACT* to the solution. Progressing to the ANN-based decisions, and with a threshold of 0.7 applied to each ANN, at the speed control stage, the network suggests that the element *PROP_DCV* be added. At the inertial control stage, the elements *CVC_A&B* and *PRV2_A&B* are deemed necessary. Since these produce no side-effects, the process moves to the final stage, the choice of contextual components. Here, none of the elements is selected in this case.

Figure 21 shows the manner in which the interface presents this solution design to the user. The solution is assessed as follows:

- *MOT_ACT* – the motor was selected directly from the specification, and so is correct.
- *PROP_DCV* – this element, a proportional valve, provides continuously variable speed, and so is correct here.
- *CVC_A&B* – this element is used principally to control some aspects of inertia in the system.
- *PRV2_A&B* – likewise, this element is used to control the effects of inertia, and is often used in combination with the previous element.

This represents a category 3 solution – the selected elements are correct, but functionality is missing from the solution (once more, there is no means of providing smooth accelerations).

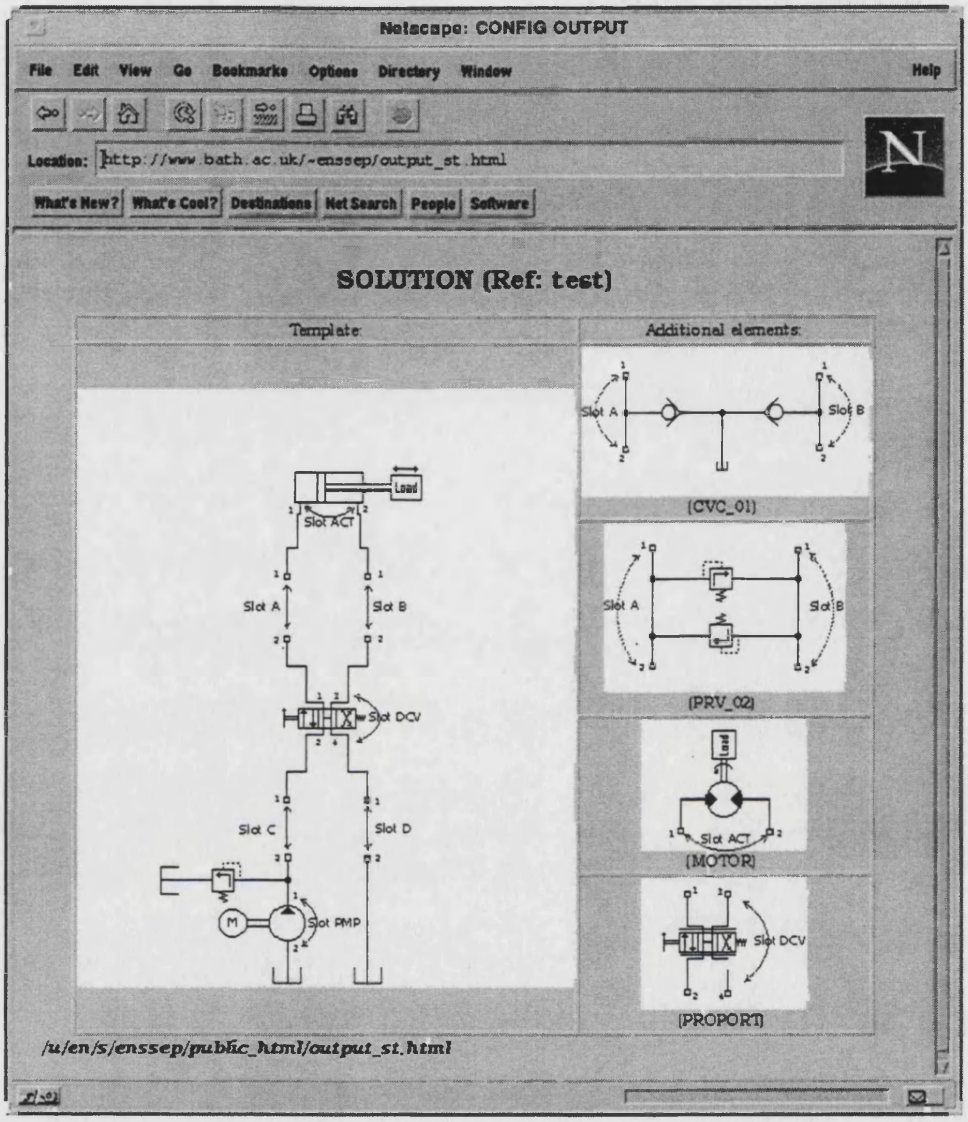


Figure 21. Browser output with suggested solution to test specification (KBS 3).

Again, the system was tested across a range of different thresholds (with a particular threshold being applied to all the ANNs in the system), and, again, the best result tended to occur in the range 0.5 – 0.7. In this range, 45 – 50% of solutions fall into categories 1 and 2 (Figure 22), the best, with 15% of solutions falling into the worst categories, 5 and 6. So, this system produces improved results when compared to the previous ANN-based system.

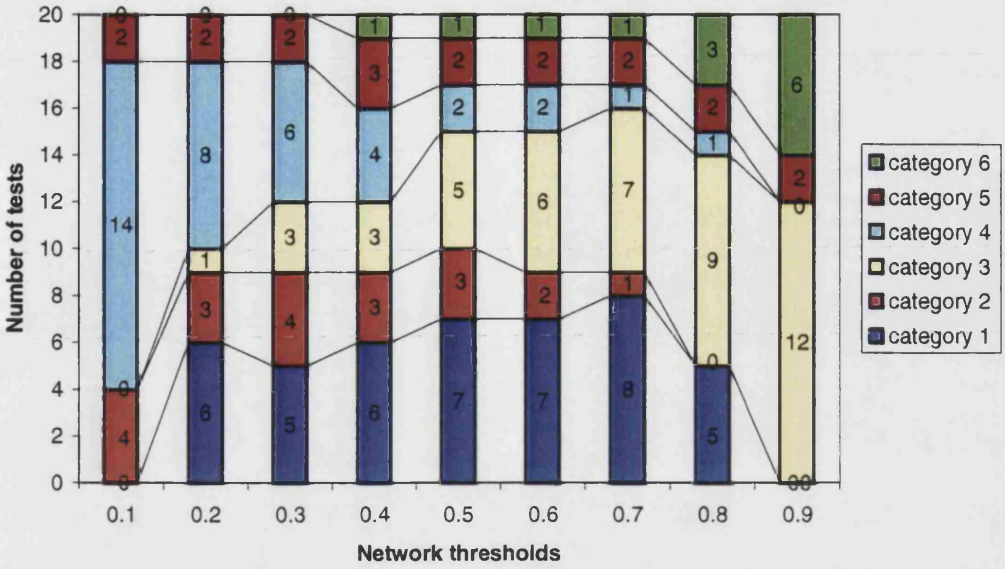


Figure 22. Multiple stage KBS test results at a range of threshold values.

7.5.7 Conclusions

In this design strategy, particular design sub-tasks are isolated, and defined solely in terms of the relevant inputs and outputs. Thus, unnecessary 'noisy' inputs and outputs are removed from the neural networks before learning begins, reducing the size of each ANN, and reducing the potential for wrong responses. On the whole, as smaller ANNs tend to model simpler relationships (the inputs and outputs are of a lower dimensionality), so they require fewer data to train adequately. Hence, the model consists of smaller, easier-to-train ANNs. However, the ANNs would seem to remain subject to the problem of under-determined weightings discussed earlier.

Any improvement of performance, however, must be seen as a trade-off against the additional heuristic knowledge that is required in the form of the design strategy. The capture of this strategy is not easy and it is subjective, being one approach to the task postulated by a particular expert; presumably, different experts may well suggest different approaches. This raises the question (which could be asked of all the systems described here) of whether data resulting from different design processes can be used to learn how to perform this particular strategy. Indeed, this leads to the wider question of whether examples from different sources display the necessary regularities and cohesion to make inductive learning of this form a viable proposition in any circumstances, regardless of the strategy adopted.

7.6 Summary

This chapter has presented three distinct KBSs for the conceptual design of fluid power circuits; for each, the required inference knowledge has been provided in the form of ANNs trained on the archive examples.

As discussed, with few examples available, it is difficult to gauge the extent to which the ANNs are learning the appropriate, generalised knowledge that is required. The usual method is to set aside a portion of the examples as a test set, and, during training, to periodically test the response of the network on this set. When the network responds to these examples in a satisfactory manner, then it is considered to be suitably trained. Here, however, there are too few examples to enable a representative test set to be formed - and, indeed, it is doubtful whether there are enough to form a representative *training* set. From the performance of the systems, and the typical numbers of training examples that are mentioned in the literature, it would seem that the networks are *under-trained.* However, due to the opaque nature of the trained networks, this remains a conjecture as to their unsatisfactory performance; as stated previously, this could also be attributed to a number of other factors – poor strategic knowledge, inappropriate representations of the domain knowledge, and so forth.

In an attempt to better assess the usefulness of inductive ML for capturing design heuristics, the following two chapters describe learning algorithms that represent their knowledge in a more comprehensible manner.

8 The Capture of Design Heuristics using a Classification Construction Algorithm

This chapter describes a KBS in which the necessary inference knowledge is represented in the form of classification rules. As in the previous chapter, this system will be described in terms of the form and content of each of its categories of knowledge, before a description of its implementation and test results.

The application of the CN2 algorithm (Clark and Niblett, 1989; Clark and Boswell, 1991), introduced in section 4.2.1, to the task of learning these classification rules was felt to be worthy of investigation for reasons which will now be outlined.

8.1 CN2 and Inference Heuristics

In response to the previous experimentation involving ANNs, the CN2 algorithm would seem to offer the following advantages for learning and representing the inference heuristics:

- Since its nature means that it tends to be used to learn 'simpler' classification knowledge than the more complex, higher-order pattern matching learned by ANNs, it might prove more amenable to learning successfully with relatively few data.
- CN2 manipulates symbolic data (that is, data in the form of labels such as *yes* or *high*). Given the qualitative nature of the developed representations, this means that the examples would require 'less' encoding to be transformed into training data, reducing the scope for making inappropriate encoding choices.
- The more comprehensible nature of the rules learned by CN2 means that the knowledge can be examined to see what is, and what is not, being learned. This can provide useful insights into a particular learning episode, into the characteristics possessed by the training data and into the nature of the learning task itself.

The principal drawback to the use of CN2 and similar algorithms in this context is the nature of the knowledge it learns, which would not appear to be 'natural' for expressing synthesis heuristics. Complex design processes cannot easily be construed in terms of classification

tasks. However, if the use of CN2 were to be investigated, these are the terms in which the task would need to be recast.

8.2 KBS 4 – *Classification Rule System*

This system adopts a design strategy which involves determining the presence or absence in a new solution of each solution element independently. The inference heuristics for this experiment, then, are in the form of classification rules – each element is classified as *present* or *absent*. To be able to apply this knowledge to new instances of design problems, these rules are embedded within an appropriate *expert system shell*. An expert system shell provides an inference engine and control mechanisms that can be applied to a user-supplied knowledge base (usually expressed in the form of rules).

8.2.1 Strategic Knowledge

Given a new design specification, the task is to select appropriate solution elements to meet that specification. This is done by considering each solution element in turn, and, based on the specification, deciding whether or not it should be present in the solution. When this decision has been made for all solution elements, then the solution is considered complete (Figure 23).

8.2.2 Inference Knowledge

The inference knowledge in this model, then, is that necessary to determine whether each solution element should be part of the solution to the current design problem.

8.2.3 Domain Knowledge

Once again, the domain knowledge is in the form of the representations of design specifications (static state representation) and solutions, and the set of values that each attribute in the representations can take.

8.2.4 Working Knowledge

The working knowledge consists of the current specification, and the status of the solution elements.

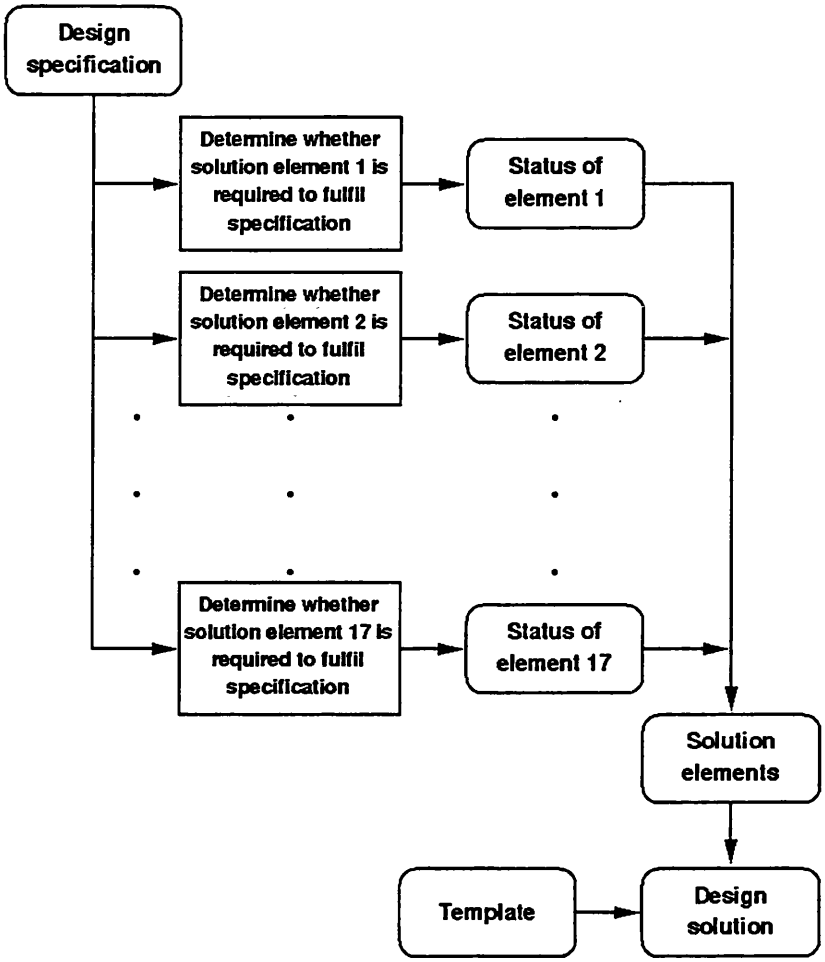


Figure 23. Classification rule-based design strategy (KBS 4).

8.2.5 Implementation – Classification Rule KBS

Given a new specification, the design strategy can be thought of as one of determining whether each solution element in turn falls into either the class *present* or else the class *absent.* There would need to be appropriate classification knowledge for each solution element – this takes the form of a set of classification rules produced by CN2. Each rule produced by this algorithm has the form:

if <condition>
then <classification>

Here, the condition would be expressed in terms of a test of some of the specification attribute values, and the classification is an indication of the presence or absence of the solution element. This model of the process is shown in Figure 24.

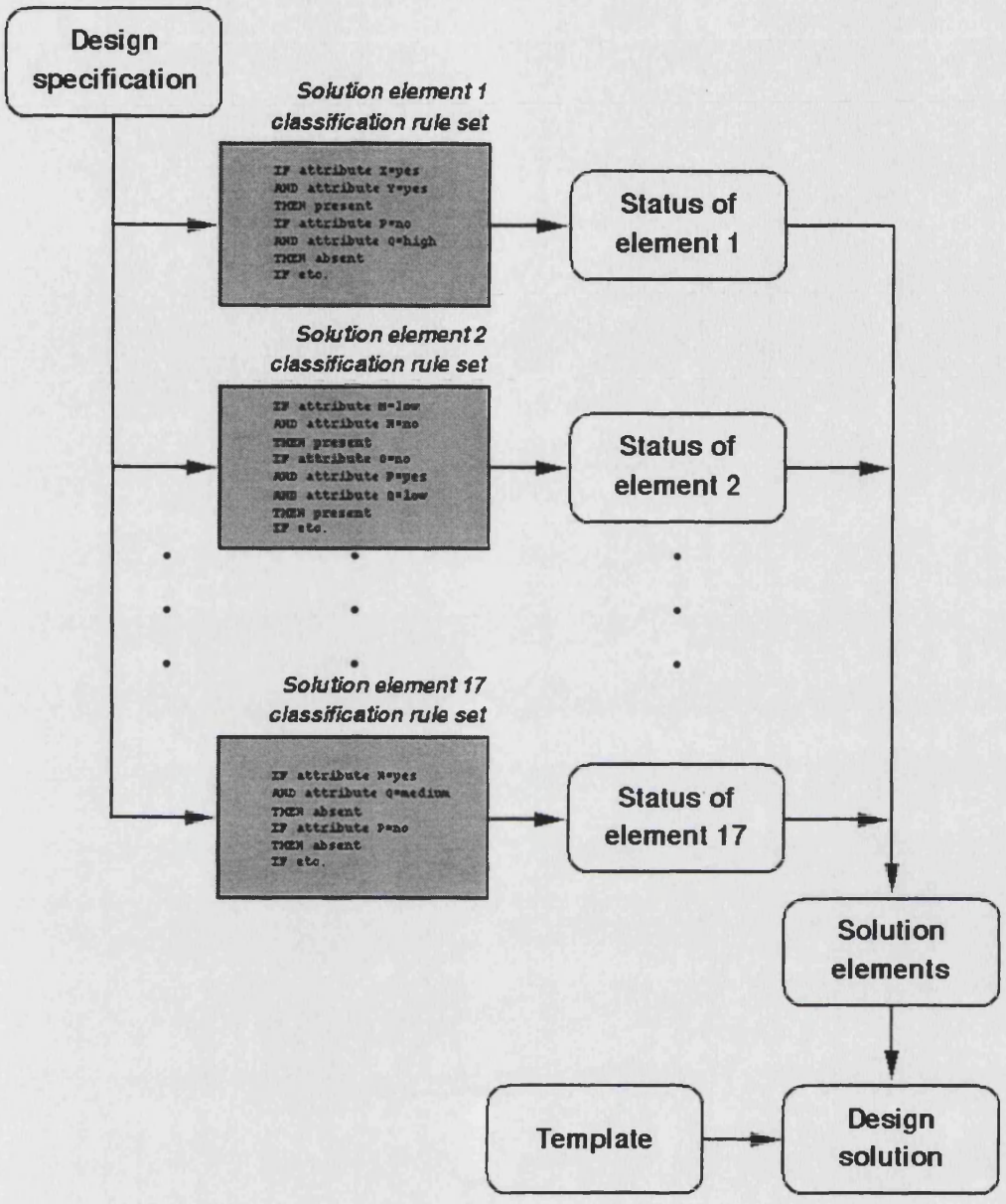


Figure 24. Implementation model of the classification rule-based approach.

Training Data

Given this model, a separate training data set is needed to learn how to classify each solution element. Hence, there would be 17 training data sets: the CN2 algorithm would be applied to each in turn, producing 17 sets of classification rules, one set per solution element.

Each training set includes data from all 30 archive design examples. Each design example is described in terms of its specification attribute values, with an additional classification attached: either *present*, if the solution element in question is present in the corresponding design solution, or else, *absent*.

Since the algorithm manipulates symbolic, discrete data values, and the specification and solution elements attributes are described in this fashion, there are fewer data encoding problems than arise when using ANNs. The values of the discrete-valued specification attributes can be described simply in terms of labels *yes* or *no*, *high* or *low*, or *high*, *medium* or *low* as appropriate. The state of the solution element in each case is indicated by the use of either the symbol *present* or else, the symbol *absent*. Each member of a training set, then, consists of 15 symbolic values, describing the specification and the class of the solution element in question in the corresponding archive example.

To illustrate this, an archive example has the specification values shown in Table 16, and the status of each of the elements in its solution is shown in Table 17. This example is represented in each of the training sets by the specification values with the addition of the appropriate status label from Table 17. So, for the solution element *POCV_A*, this example would give the training pattern shown in Table 18. The other 29 archive examples, described similarly, completes the training data set for this particular solution element.

attribute name	*value*
maximum load/force	low
maximum speed	medium
plane of motion	non-horizontal
continuously variable speed	no
hold load stationary	yes
smooth accelerations	no
hold load on failure	yes
load-independent speed	no
control extend speed	no
control retract speed	no
solution requires motor	no
control inertia	yes
energy efficiency paramount	no
control accuracy	low

Table 16. Example archive specification.

The choice of learning parameters is also easier for CN2 than for ANNs. A parameter governs the size of CN2's 'working memory' which stores potential rules. A larger memory results in a greater time required for learning, but potentially more effective rules. Here, as training times are of no consequence, the learning being 'off-line', the maximum setting permitted by the CN2 implementation used (Boswell, 1990) was allocated to this parameter.

solution element	*class*
PRV1_A	**present**
PRV1_A&B	absent
POCV_A	**present**
CBV1_A	absent
CBV1_A&B	absent
CBV2_A	absent
PCMP_C&PROP_DCV	absent
DECV_A&B	absent
MO_A	absent
MO_B	absent
CVC_A&B	absent
VPCRV_D	absent
PRV2_A&B	absent
VDP_PMP	absent
MOT_ACT	absent
PROP_DCV	absent
CC_DCV	absent

Table 17. Class information for archive example.

The algorithm was applied to the training data and, accordingly, 17 rule sets were created. As an example, the rule set generated for the solution element *POCV_A* is as follows:

```
IF      hold_on_failure = no
THEN    element POCV_A = absent [18]

IF      smooth_accelerations = yes
THEN    element POCV_A = absent [7]

IF      motor_required = yes
AND     energy_efficiency_paramount = no
THEN    element POCV_A = absent [7]
```

```
IF      smooth_accelerations = no
AND     hold_on_failure = yes
AND     motor_required = no
THEN    element POCV_A = present [6]

IF      smooth_accelerations = no
AND     max_load = high
THEN    element POCV_A = present [1]
```

attribute name	*value*
maximum load/force	low
maximum speed	medium
plane of motion	non-horizontal
continuously variable speed	no
hold load stationary	yes
smooth accelerations	no
hold load on failure	yes
load-independent speed	no
control extend speed	no
control retract speed	no
solution requires motor	no
control inertia	yes
energy efficiency paramount	no
control accuracy	low
class	present

Table 18. Example training pattern for solution element POCV_A.

This element is generally used to hold a load in position, especially in response to system failure. The rule conditions test the values of specification attributes, and the conclusions indicate whether or not, on the basis of these values, this element should be present in the solution. Attached to each rule is an indication of the number of archive examples that the rule 'explains' - the greater this number, the greater should be the confidence that the rule is a good one. This information is later used in during the design process: if more than one rule for a certain solution element is found to be applicable given the new design specification, then the rule chosen for application is that in which there is the greatest confidence.

The classification rules were placed within an inference engine along with appropriate control structures to form the KBS (the *CLIPS* expert system shell (Giarratano and Riley, 1997) was that used in this instance). This shell handles the 'firing' of rules (ensuring that a maximum of one rule fires for each solution element), the resolution of conflicts between several applicable rules, etc., and in so doing, applies the overall design strategy. A new specification is supplied in the form of a set of initial 'facts' to the shell, from which it attempts to infer all that it can from the rules in a *forward chaining* manner (in other words, the process is data-driven – the arrival of the specification data prompts the system to infer the status of the solution elements).

On presentation of a new design problem, the 'best' applicable rule for each solution element is found. This is the rule having a condition that is satisfied by the current values of the specification attributes. In the event of several rules meeting this criterion, that which explains the greatest number of the training examples is selected. This rule is then used to determine the status of the element in the design solution. In this way, the description of the solution is generated, consisting of those elements classified as being present. If no applicable rule is found for a particular solution element, then the status of the element remains undetermined at the completion of the inference process (and hence, the design process remains uncompleted).

8.2.6 Results

Returning, once more, to the test specification given in Table 15, this was provided to the system as it appears in the second column of the table, with no need for encoding. For the element *PROP_DCV*, the rule:

```
IF      continuously_variable_speed = yes
AND     load_independent_speed = no
AND     energy_efficiency_paramount = no
THEN    element PROP_DCV = present [7]
```

was invoked, adding the element to the solution. Similarly, for the element *DECV_A&B*, the rule:

```
IF      smooth_accelerations = yes
AND     hold_load_on_failure = no
THEN    element DECV_A&B = present [4]
```

was fired, adding this element. Finally, for the element *CVC_A&B*, the rule used was:

```
IF      control_extend_speed = no
AND     plane = horizontal
THEN    element CVC_A&B = present [5]
```

and the element was selected. For all other elements, rules were fired that deemed them to be *absent* from the solution.

Hence, the solution contains the following elements:

- *PROP_DCV* – used to provide continuously variable speed, and so, is correct here.
- *DECV_A&B* – used to provide smooth accelerations; again, correct.
- *CVC_A&B* – used to control inertia, another correct element.

Since the selected elements are all correct, and all the desired functionality seems to have been provided, this represents a category 1 solution.

An additional aspect of this approach is the opportunity it presents to investigate the criteria upon which the elements have been selected or rejected. Here, the conditions of the rules for *PROP_DCV* and *DECV_A&B* refer to *continuously variable speed* and *smooth accelerations* respectively, and so, from what is known of the domain, these rules would seem to have some credibility as knowledge of the task. On the other hand, the rule conditions for *CVC_A&B* make no mention of *control inertia*, as might be expected, and so, while the choice is correct, it appears to have been based on spurious knowledge. However, this rule might indicate that the algorithm has detected a hitherto unnoticed (and valid) piece of knowledge about the task (and one that is supported by the evidence of 5 examples).

The performance of this system, in general, seems better than those produced through the ANN-based approaches. With no need to investigate thresholds for the rule-based system, the results indicate that 55% of solutions fall into category 1 (Figure 25). It should be noted that no results fall into category 2 – the rule-based system has fewer tendencies to produce decent but over-engineered solutions than do the previous systems. Only one of the solutions falls into either of the worst two categories.

While the same difficulties of testing the knowledge apply here, unlike the trained network, the rules are comprehensible, and can be examined to see if they represent good (or, at least, plausible) design knowledge. Again, the lack of data is evident in the performance. Some of the rule sets are not general enough to cope with all situations. Some new combinations of specification values do not match the conditions of any of the rules in a set; hence, no rule is applied, and the status of the corresponding solution element in the final design remains undetermined. However, this state, which corresponds to a lack of synthesis knowledge, is preferable to the response of an ANN in a similar situation. Rather than indicating its ignorance by not making a decision, as is the case here, the ANN applies its incorrect knowledge (unmodified weightings) to calculate its response — in effect, it 'guesses' what the output should be. The CN2 algorithm, on the other hand, assumes no such initial knowledge of the task.

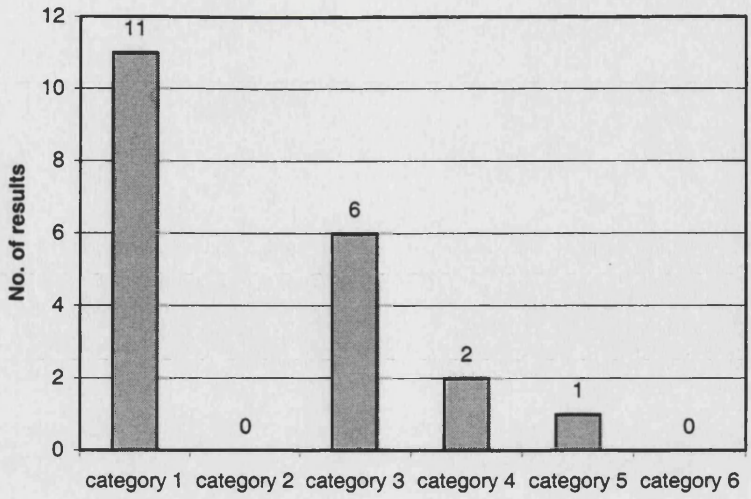


Figure 25. Results of testing the classification rule KBS.

The algorithm does learn some 'poor' rules, which, when applied, produce inappropriate solutions. The measure of confidence CN2 attaches to each of the rules it generates is of some use here, since these poor rules tend to be supported by few of the examples. However, with 17 solution elements and only 30 examples, there are often few instances of, in particular, the presence of a certain element in a solution, and so, the rules indicating that the element is *present* will rarely be supported by a great deal of evidence.

8.2.7 Discussion

The design strategy applied in this model would not seem to be a natural reflection of the process, and as such remains unsatisfactory. Design solution synthesis would seem to require a more 'holistic' approach than the piecemeal consideration of solution elements. However, describing the process in terms of a series of independent classification tasks, and thereby simplifying the individual learning tasks, may represent a practical approach for using inductive ML to build a KBS for complex tasks for which few examples are available.

As described above, the CN2 algorithm makes an initial 'closed world' assumption. The algorithm begins with no rule knowledge about the task, and only when explicitly induced are rules added to this knowledge. Hence, the knowledge learned by the algorithm comprises only valid rules, inasmuch as they are supported (to a greater or a lesser extent) by the evidence in the examples. This contrasts with operation of an ANN: the random initialisation of parameter values, instils certain (in all probability invalid) 'rules' into the network at the outset of learning - it is assumed that these will be modified into valid 'rules' during the training phase. However, if there are few data, many of these will not be adequately

modified, and if invoked in subsequent use, may well have a detrimental effect on the quality of solutions.

The fact that the rules are not general enough to classify all solution elements in every instance makes apparent a trait of the learned knowledge suspected, but hidden, in the trained ANNs: namely that the learned heuristics are not complete enough to cover the range of possible problems. There are design specifications that cannot be translated in their entirety into appropriate solution 'causes' using this knowledge, and so, the design process remains unfinished.

This approach offers an additional advantage in that the learned knowledge is comprehensible in a way that a trained neural network could never be. This means that appraising the quality of the learned knowledge can be made independently of the appraisal of the system in which it is embedded (by, for instance, showing the rules to domain experts), and perhaps even corrected as necessary. Certainly, many of the learned rules do seem to be reasonable in the domain. However, with the need to reverse engineer certain of the archive examples, and describe all at the level of maximal functionality (mentioned in section 5.3.1) comes the suspicion that the learned rules are merely echoing the sort of relatively simple analytical knowledge that was applied in doing so.

As an aside, the ILP algorithm GOLEM (section 4.5.1) was also applied to this classification learning task - in this case, the task was to learn the logic 'program' that would allow the correct classification of solution elements. The resulting knowledge was found to be very similar to the knowledge expressed by the CN2-generated rule sets. Since the representation of the specification is in the form of attributes that are considered to be independent, the relational learning aspect of ILP approaches, their chief attraction, is not exercised.

8.3 Summary

In the KBS presented in this chapter, the inference knowledge is represented in the form of classification rules. These have been generated from the archive examples by the CN2 algorithm.

In contrast to the ANN-based approaches discussed in the previous chapter, this knowledge is comprehensible and can be examined independently of the system itself. The 'gaps' in this knowledge, instances when no rule is applicable to the current specification, make evident that which is suspected in the behaviour of the ANNs, namely that the learned knowledge is insufficiently general to cope with all design problems.

The improved quality of the solutions generated by this KBS might be a reflection of a number of simplifying assumptions made in producing the data and developing this model of the design process. That said, this approach would seem to offer a more practical approach to the generation of inference knowledge from few examples, perhaps with the assistance of a design expert. The symbolic nature of the rules encourages their correction, and rules based on little supporting evidence can be treated and used with due caution. This sort of 'semi-autonomous' approach to acquiring heuristics would seem to be worthy of further investigation. However, it is beyond the scope of the research documented in this thesis.

9 The Capture of Design Heuristics using a Conceptual Clustering Algorithm

This chapter presents the final KBS developed using ML to be discussed in this thesis. In this case, the algorithm used is COBWEB (Fisher, 1987), a conceptual clustering algorithm (section 4.3.1). Before describing this system, the rationale suggesting such an approach will be summarised.

9.1 COBWEB and Inference Heuristics

The use of a conceptual clustering algorithm for capturing design synthesis knowledge is based on the following rationale:

- COBWEB produces a conceptual hierarchy, based on similarities among examples, which has been suggested by Reich and Fenves (1992), amongst others (see section 4.9.1), as being useful for implementing a case-based reasoning-type approach to design. This provides the motivation for this investigation of the approach.
- Again, the algorithm manipulates discrete symbolic data, and so may be more appropriate for learning about this task, given the manner in which it is represented here.
- The algorithm works incrementally: new examples can be added to the existing hierarchy when they become available. As such, the algorithm can incorporate new examples as and when they arrive, and so, might be thought to be better suited to the 'on-going' nature of design.

9.2 KBS 5 – *Conceptual Hierarchy System*

In this system, a conceptual hierarchy of design specifications is, in effect, used as the similarity metric in a CBR approach. The CBR approach to design relies on the existence of some 'memory' of previous design episodes. When a new design problem is encountered, the strategy is one of recalling the episode involving a specification most similar to the new problem; the solution produced during this episode is then proposed as the new solution. However, unless the design memory is relatively comprehensive, it is unlikely that this

solution will be adequate in its retrieved form – in which case, the solution will need to undergo some modifications in order to be acceptable. (This modification phase is not implemented in this KBS.)

9.2.1 Strategic Knowledge

So, the design strategy is one of retrieving that design solution from the archive that meets a design specification that is 'most similar' to the current one – this is proposed as the solution to the current problem. This approach is illustrated in Figure 26.

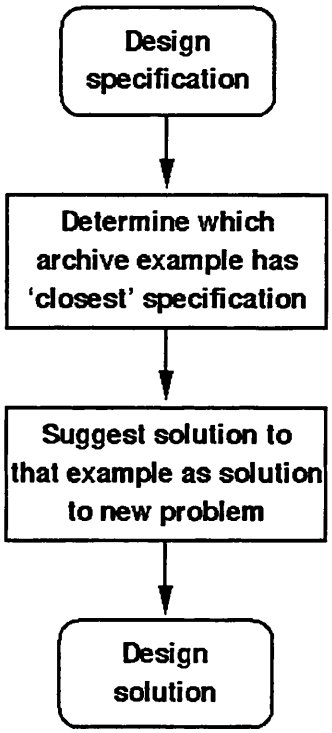


Figure 26. Conceptual hierarchy system design strategy (KBS 5).

9.2.2 Inference Knowledge

The inference knowledge necessary to implement this strategy consists of the knowledge of how to determine which archive example has a specification most similar to the current one.

9.2.3 Domain Knowledge

The domain knowledge is in the form of design specification attributes and their values, described according to the static state representation, and the set of archive examples, each of which consists of a complete specification and a corresponding complete design solution.

9.2.4 Working Knowledge

The working knowledge consists of the current specification, and the best-matching example.

9.2.5 Implementation – Conceptual Hierarchy KBS

The inference knowledge required to implement this model can be expressed in the form of a conceptual hierarchy of the sort learned by COBWEB. A hierarchy is formed of the archive examples, based upon the description of their specification values alone. A new specification can be classified (using the category utility measure) through this hierarchy, eventually being classified into one of the terminal nodes, which will correspond to the description of the specification of a particular archive example. As it has been judged on this basis to be similar to the new problem, the design solution of this example is proposed as the solution to the new problem. Hence, the machine learning algorithm has learned the similarity metric that is applied. Figure 27 shows this design system implementation model.

As such, this represents only a partial implementation of a CBR system – since relatively few archive examples are available, a more complete system might go on to instigate an adaptation mechanism by which the retrieved solution could be made to satisfy 'more nearly' the specification.

Training Data

So, the COBWEB algorithm is used to learn this hierarchy. It is an unsupervised algorithm, searching for similarities implicit within the descriptions of examples, rather than relying on explicit output or class information. The training data from which the hierarchy was to be inferred consists of 30 archive examples: each is described in term of its specification attribute values alone (so, for example, Table 16 shows how a pattern is represented in the training data). In addition, a label is attached to each example, to indicate its reference number in the archive: this number is used merely to label the terminal nodes in the hierarchy, and so indicate which solution would need to be retrieved, and is not used during the construction of the hierarchy itself. Since the algorithm operates on qualitative, symbolic data, no further encoding of the examples was required.

On the basis of this data, a conceptual hierarchy was produced using an implementation of COBWEB (Mooney, 1991). This hierarchy is shown in Figure 28 (also shown in this diagram are the probabilistic descriptions of two of the concepts within the hierarchy - one is a terminal concept, corresponding to one of the archive design examples; the second is more

general, describing the concept formed by 8 of the design examples). This hierarchy was embedded within appropriate code to accept and classify a new specification, and indicate the reference number of the suggested solution, thus producing a partial CBR-type design system.

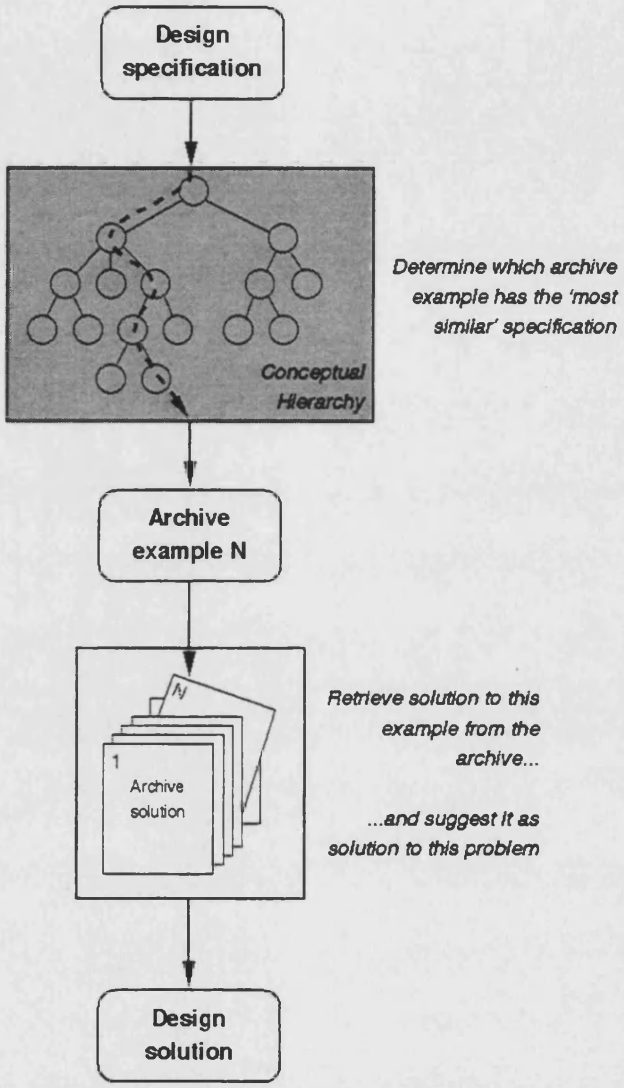


Figure 27. Implementation model of conceptual hierarchy system.

9.2.6 Results

No testing of this system was attempted: see below for a discussion of the reasons for this.

9.2.7 Discussion

The generated hierarchy can be used as described above to classify a new specification, and retrieve a solution design accordingly. This has the advantage of returning a complete, fully

configured fluid power circuit. However, given the limited amount of examples, it is unlikely that the returned design will meet the specification in its entirety.

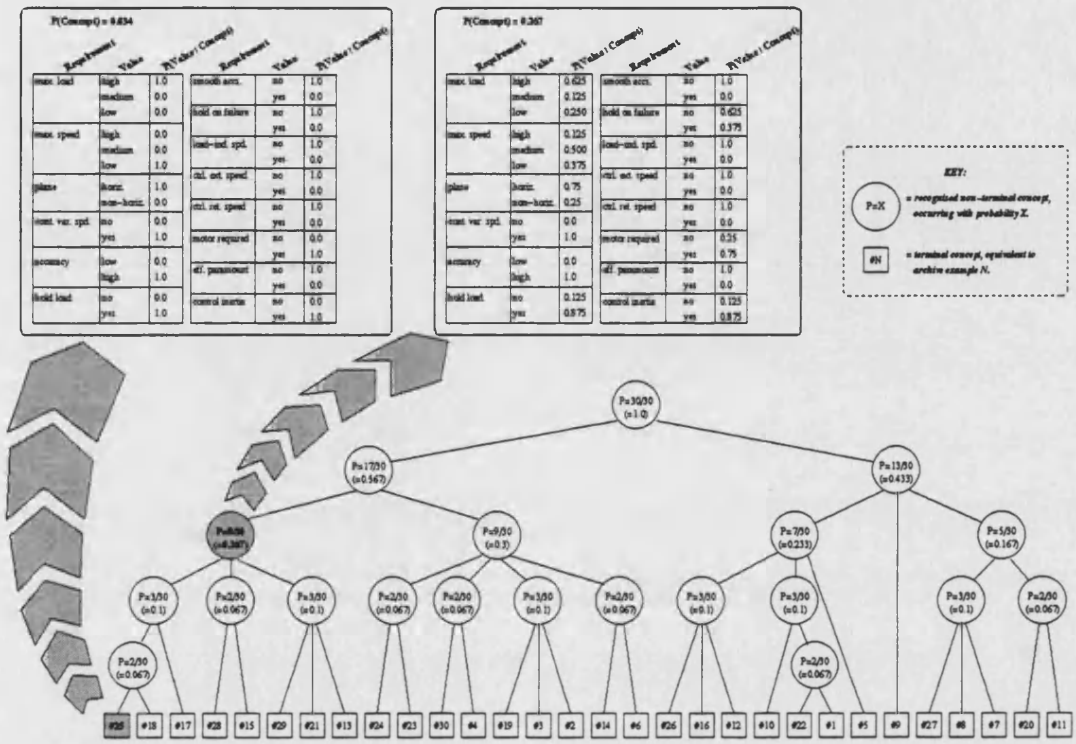


Figure 28. The learned conceptual hierarchy of design examples.

However, while this might seem an attractive approach, a major theoretical objection can be raised against such a method. Since it forms the classification hierarchy based solely upon the specifications of the available design examples, this hierarchy merely reflects the similarity patterns amongst the (limited number of) specifications found in the archive. It cannot be said to be a hierarchy *for design*, since it incorporates no information about the subsequent use of the specifications during the design process. Presuming some modification of the retrieved solution is going to be needed, a sensible strategy would be to ensure that those attributes that have greatest influence over the whole character and choice of solution elements are already satisfied in the retrieved solution. These attributes would, presumably, be more difficult to satisfy through later modification of a solution, as this would involve global, rather than local, changes. A lesser importance would be attached to matching those attributes that can be more easily satisfied later by, say, the simple addition of an element, with no detrimental effects on solution elements already in place.

The previous ML approaches used were provided with some concept of the 'purpose' of the specifications, inasmuch as they employed supervised learning, and so the data included the

corresponding solution elements, in addition to the specification attribute values. The most that can be said about the produced hierarchy is that it reflects the similarities and biases amongst the specifications of the available archive data.

For this reason, there are no grounds for believing that any hierarchy generated in this manner corresponds to the inference knowledge required to implement this design strategy. This, and the lack of a modification aspect to 'complete' the design task, meant that no methodical testing of this system was attempted, and the use of conceptual clustering techniques such as COBWEB were investigated no further.

9.3 Summary

Section 6.2.3 emphasised the importance of using inference knowledge that is appropriate in both form and content – in this case, although it has the right form, the content of the conceptual hierarchy is not that required to implement successfully the strategy. This shortcoming has been overlooked by the proponents of this approach in the past.

The approach remains an appealing one, though. As mentioned in section 3.6 above, the need to acquire an appropriate similarity measure is one of the disadvantages of the CBR approach. If some ML algorithm could supply this, this difficulty would be circumvented. However, none of the algorithms currently available would seem to be suited to the task of learning this knowledge.

10 The Capture of Design Heuristics using Inductive Machine Learning

This chapter begins with a summary of the issues that have been raised in the course of the development of the ML-based KBSs described in the previous chapters. This is followed by a comparison of the results of the testing performed on some of the KBSs, and the chapter concludes with an evaluation of the usefulness of inductive machine learning for the acquisition of design heuristics.

10.1 The Development of ML-Based Design KBSs – Issues

The process of constructing these KBSs raises a number of general issues surrounding the application of inductive ML to capture design heuristics, and the subsequent use of this knowledge.

10.1.1 The Strategic Knowledge

The nature of the available design information that has been collected into the archive would seem to exemplify the *inference*, rather than the *strategic* heuristics, in that it illustrates the relationships between specification attributes and solution elements, but not the high level processes by which these relationships are employed to solve design problems. Hence, the archive can be used to try to learn inference knowledge, but the strategic heuristics for the KBSs must be derived from some other source.

Consequently, the strategic knowledge must be either knowledge-engineered or handcrafted, and since this is heuristic knowledge, acquired through experience, and difficult to express, neither approach seems particularly satisfactory. The only way to ascertain that the strategy is a valid one is through its use in the construction of a complete, successful KBS. This has not been possible here, so the strategies adopted must retain the status of hypotheses about this particular design process.

When developing these strategies, consideration must be given to the particular forms of knowledge that the available ML algorithms can learn. In order to make use of a particular

algorithm to learn inference knowledge, it is necessary to have a design strategy in which inference knowledge of the form learned by the algorithm can be used to generate solutions. As a consequence, the intention to use ML algorithms can unduly influence the strategy (as may well be the case in the classification rule-based approach seen in chapter 8, for example), with the result that the strategy seems remote from the actual process. However, this sort of compromise might be necessary if design KBSs are to be constructed.

10.1.2 Encoding the Training Data

The training examples must be presented to a ML algorithm in a form appropriate to that algorithm. This can mean that the archive examples must be re-described before they can be used with a particular algorithm. This must be done in such a way as to preserve the information contained within the examples as far as is possible. This is not a trivial task, with a number of competing encodings available, and often little indication of which is the best. A bad choice of encoding can make the learning task more difficult or even impossible; however, picking a good encoding often requires a certain amount of knowledge about the problem in hand and about the characteristics and operation of the algorithm.

Given the nature of the representations developed for this problem, which (with the exception of the temporal state specification representation) are qualitative in nature, the principal encoding difficulties were encountered here when using ANNs. Presumably, however, similar problems would be met when attempting to convert quantitative values for use with symbolic learning algorithms.

10.1.3 ML Algorithm Training Parameters

Before training, suitable training parameters must be supplied to the ML algorithm. Selecting the proper parameters can be a difficult task, and often relies on a trial-and-error approach of repeated attempts to arrive at some satisfactory state.

Again, this problem seems to be particularly acute in the case of ANNs, which have a number of such parameters: the number of hidden layers, the number of units in each hidden layer, the learning rate, the number of repeated iterations through the data, and the function applied by a unit to its input.

10.1.4 Testing the Learned Knowledge

The conventional manner of determining whether an ML algorithm has successfully learned some body of knowledge involves interrupting the training process periodically, and applying the knowledge to a separate set of test data. When the knowledge provides a

satisfactory response to this test set, then the desired knowledge has been learned and training is complete.

However, this approach was not applied here for the following reasons. First, and foremost, with few examples available, any subset devoted to testing could not be considered representative enough to provide an accurate measure of the quality of the learned knowledge. Consequently, all of the available examples were included in the training sets. This would seem to be a particular difficulty when training ANNs, since the networks are initialised with random weightings and so the outcome can be quite different from one training episode to the next, more so than for the other classes of inductive algorithm. As a result, the quality of the learned knowledge can vary greatly, being dependent on this initialisation, and so, some manner of gauging this quality would seem necessary. Furthermore, it is not possible to measure the extent of the learned knowledge by any means other than this sort of testing (whereas it is possible to inspect and understand the heuristics generated by the algorithms that express their knowledge in a symbolic form). This difficulty suggests that there are simply too few examples available for these inductive ML approaches.

In addition, since several valid design solutions can fulfil a particular specification, comparing the generated solution with the suggested one for a test case can be misleading. This is perhaps indicative of a more serious flaw in this approach, as will be discussed later.

Consequently, the algorithms were trained to give satisfactory performance on their training data, and then incorporated into the KBS, which was to be tested.

10.1.5 Testing the Design KBSs

A related issue, then, is the testing of the systems themselves. For the reasons outlined in the previous section, the automatic testing of systems was discounted. Furthermore, rather than simply labelling an unsatisfactory solution as 'incorrect', some indication of the degree to which it fails to meet its specification is of interest in the context of this research.

As a result, there would seem to be two approaches to testing systems: first, by building the suggested solutions, and comparing the performance against the specification. Secondly, by 'manually' appraising the suggested solutions. The first approach was not practical in these circumstances. Consequently, the second approach, though difficult and time-consuming, was that adopted here. However, to be able to assess the quality of solutions requires a certain amount of expertise of the domain.

For design KBSs that work less than perfectly, it does not seem possible to provide any absolute measure of their 'goodness'. Consequently, the method adopted (section 6.3.1) provides some measure of the relative worth of the systems.

10.1.6 The Opacity of the Learned Knowledge

As mentioned, the knowledge learned by ANNs is virtually incomprehensible (it takes the form of a network of weighted connections). In design contexts, and given the difficulties of testing, it would be useful to be able to examine the sorts of relationships that are being learned, and those that are not. Algorithms such as CN2 might be less appropriate to the task in terms of the form of knowledge they produce, but the intelligibility of their output can make them more useful for investigating the adequacy of data and the nature of the learning task itself.

The production of 'comprehensible' knowledge also offers the possibility of manually modifying the heuristics so as to improve the performance of the KBS. This may present a practical approach to the construction of design KBSs, using partially correct heuristics as a basis for and stimulus to knowledge acquisition from a human expert.

An additional consideration is that the use of, for example, rules in a design system means that some sort of explanation of the choice of solution elements can be given, in terms of the values of attributes that have influenced the choice. It has been suggested that such explanations may be necessary if these systems are to gain acceptance as useful tools outside academe (Rich and Knight, 1991).

10.1.7 The Nature of the Learned Knowledge

The nature of the knowledge generated by the ML algorithms when trained upon relatively few data is of relevance here. In the case of ANNs, the outcome of the training might well be an 'under-trained' network, with some connection weightings remaining inadequately modified. However, this state is only apparent in the performance of the network: nothing else suggests that it may be under-trained.

In the case of CN2, rules are only generated in response to explicit evidence in the examples. With few examples, this can lead to gaps in the knowledge, with no heuristics available to deal with certain combinations of specification values. Hence, some indication of the 'ignorance' of the knowledge is evident. Furthermore, associated with each classification rule is a value indicating the number of training examples that the rule 'explains'. This is useful information; a greater degree of faith should be placed in any inductive knowledge that is based on a greater number of observed examples.

If, as seems likely, it is overly optimistic to expect complete, accurate inference heuristics to be generated by the ML algorithms, the sort of knowledge learned by CN2 would seem to be more useful, offering, as it does, some indication of where knowledge is lacking, and where it is based upon little corroborating evidence.

10.1.8 The Content of the Learned Knowledge

If a particular strategy is to be successfully implemented, then it is necessary to have the precise inference knowledge that the strategy demands. As seen in chapter 9, while the inference heuristics might appear to have an appropriate form, consideration of the behaviour of the algorithm can suggest that the content of this knowledge is wholly unsuited to implementing the strategy. In this case, the clustering algorithm COBWEB produces a conceptual hierarchy, and as such, is able to select the archive example having a specification most similar to a new design problem. However, the notion of 'similarity' embodied by this hierarchy is not that required to implement the CBR-type strategy for design.

10.2 Comparison of KBS Results

By way of a summary of the results, it is possible to present a comparison of the single stage ANN-based KBS (section 7.4), the multiple stage ANN-based KBS (section 7.5) and the classification rule KBS (section 8.2) (Figure 29). The ANN-based systems are shown at 'good' threshold values, 0.6 and 0.7 respectively. It can be seen that the ANN-based applications perform less well than does the classification rule-based system, in terms of the number of solutions falling into categories 1 and 2. All systems produce similar numbers of category 3 and 4 solutions, while the single stage ANN KBS produces a good deal more 'bad' category 5 and 6 solutions.

As indicated earlier, the introduction of additional task knowledge, in the form of a more detailed design strategy, into the multiple stage ANN KBS is reflected in an improvement in performance when compared to the single stage ANN-based system.

However, the classification rule-based system out-performs both. There are a number of factors which may contribute to this. In the strategy employed in this KBS, the process is divided into a series of discrete classification tasks, which, although it is not particularly natural way of thinking of design, does mean that the individual learning tasks are simpler. Hence, they require fewer examples to learn, and so the strategy may be better suited to environments in which a small number of data are available. In addition, the method of

reverse-engineering the data might have left its own traces. Because this was done by a non-expert in this design task, there may well have been a tendency to complete the examples using simple rules - precisely the sort of simple rules that are learned by CN2.

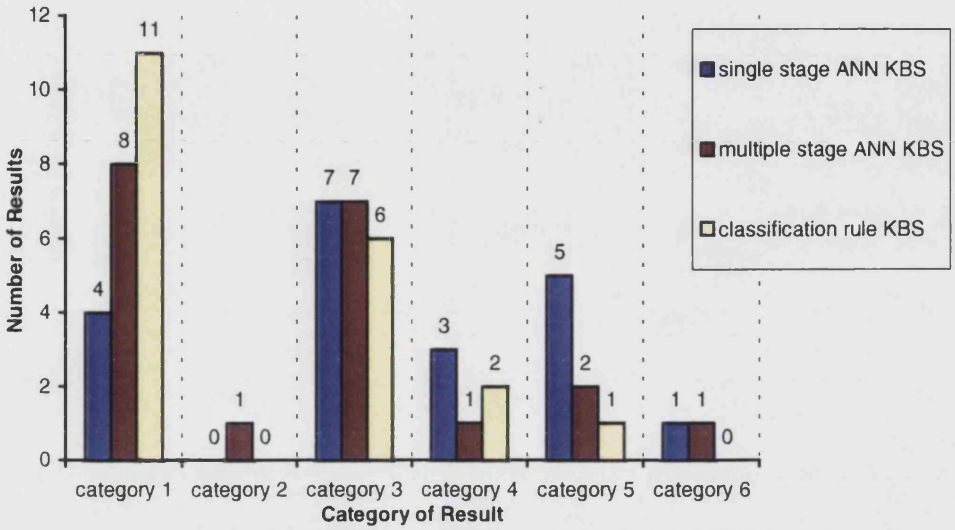


Figure 29. A comparison of the test results of the KBSs.

Overall, however, the results generated by these KBSs are disappointing. The testing suggests that little over half of the solutions generated by the best of the systems can be considered to be 'good', in so far as they seem to provide all the necessary functionality, and none that is not required. (It should be remembered, though, that this appraisal of solutions has been performed by a non-expert, and so the extent to which these 'good' solutions embody actual expertise of the task remains open to question.) A design KBS operating at this level of performance would be of limited usefulness in practical situations.

10.3 Machine Learning Design Heuristics – Discussion

The issues surrounding this approach and the results of the developed KBSs give rise to a number of grave doubts about the appropriateness of attempts to machine learn inference heuristics knowledge inductively. These doubts surround the nature of the training data that are available, and of the ML algorithms themselves.

10.3.1 The Nature of the Training Data

Many of the misgivings about the data have been expressed in section 5.3. To summarise, problems lie in:

- *the number of examples, and the representativeness of the archive* – to learn sufficiently generalised knowledge about any problem in an inductive fashion, a certain amount of data, suitably representative of the domain, must be available. The results suggest that not enough data are available in this case - and this might well be a problem encountered in any 'real' design context.
- *the description of the examples* – each example has to be represented in such a manner as to express the information essential to the task. For design problems, the production of these representations is not an easy task, and necessarily involves some degree of approximation.
- *the incompleteness of the examples* - example specifications are often incomplete, and require some degree of reverse-engineering if they are to be useful. This involves some interpretation of the data, introducing the possibility of misinterpretation.
- *the quality of the examples* - the quality of a solution is subjective. A poor solution may not be apparent as such. Learning from examples of poor quality may be reflected in knowledge of poor quality.
- *the expression of the examples* - the subtleties of design specification expression can often be lost. This can lead to problems when attempting to learn from the data, with the apparent loss of causal relationships between specifications and solution elements. Attempts to restore this can lead to the introduction of errors.
- *the style of design* - differences in designer styles and the evolution of the domain can render a set of design examples inconsistent, and hamper any attempt to learn from them.
- *the type of design* - the incorporation of anything other than examples of routine design into the archive may hinder learning.

10.3.2 The Nature of the ML Algorithms

The inductive ML algorithms that are currently available are unsophisticated. Typically, these algorithms have been developed and tested using learning tasks of a complexity less than that of learning design inference heuristics. To learn successfully, these algorithms generally expect data to be the product of a consistent, invariant system, with easily

identifiable inputs and outputs. The data should be available in sufficient quantity to allow generalisation across the range of possible inputs. The effectiveness of these algorithms in learning tasks that do not possess these characteristics is doubtful.

Design problems would appear to share few of these features. The design examples that have been collected are from a variety of sources and consequently, have been developed from different designers. So, although apparently examples of the same task, they are unlikely to be products of the same design process. Design strategies would seem to be heuristic and subjective in nature. Designers develop different understandings of the relationships between specification and solution and different designers can produce different, equally valid, solutions to the same design problem. The domain is not consistent; new products are introduced, new practices are encouraged and design solutions can be found to have serious flaws after years of service. The task cannot be easily circumscribed, if at all, with each of the relevant inputs and outputs identified. There are relatively few data available, with those that are often poorly documented. Rather than being some objective, mechanistic procedure, design is a process performed by humans for humans, with all the nuances, subtleties and caprices that this entails.

While this might paint a bleak picture for the future of ML algorithms in such applications, it should be remembered that human designers manage to learn to perform design in similar contexts. The emphasis in ML research to date has been placed on the development of general-purpose, domain-independent algorithms. The difficulties encountered here seem to demand design-specific (and perhaps also domain-specific) learning algorithms, able to cope with the particular character of available design examples, and better suited to representing design problems. Rather than attempting to learn in a vacuum, unaware of the domain beyond the description of the examples, such algorithms would have recourse to other, analytical knowledge of the domain to guide their learning, as well as access to wider 'common-sense' knowledge.

This is not to say that these, and the earlier, experiments in automatically learning synthesis knowledge are worthless exercises. Design is a complicated endeavour, and it is evident from the literature that it is, as yet, poorly understood. It is necessary to learn about the task, to comprehend more fully its character, before proposing solutions to the problems that arise. Many of the issues raised here (and, no doubt, there are many more, both general to design tasks and specific to particular problems) are not immediately obvious, only becoming apparent when an attempt is made to apply machine learning to find solutions to real-world problems.

10.4 Summary

This chapter and the previous four have been devoted to an investigation into the suitability of machine learning techniques to capture the inference heuristics that will allow a design specification to be translated into the conceptual design of a fluid power circuit. This investigation is based upon the construction of a number of design KBSs, with, in each case, the necessary inference knowledge provided by the application of inductive ML to a set of examples of this design process.

The experimentation with these KBSs indicates that by this approach some simple inference knowledge can be acquired – the examples given earlier in section 8.2.6 of the rules produced by CN2 seem, on the whole, to represent plausible, general heuristics about the selection of that particular solution element. However, these rules would intuitively seem to be far from the most appropriate representation of such heuristics, considering, as they do, a single element in isolation from the others. The representation of the ANN-based approaches, with the simultaneous choice of a number of elements in response to consideration of a number of specification attributes would seem to be much nearer to expressing the sort of heuristics that might be used by a human design expert.

In general, the learned heuristics fall far short of the ideal. In the first place, they are not always correct – poor solutions are generated by all the developed systems. The heuristics are not selecting the 'best' solution hypotheses in these cases – the data has allowed the generation of incorrect heuristics. Secondly, the heuristics are incomplete: they are either too few, or not general enough to permit a correct response to all possible specifications. This is shown most clearly in the case of the classification rule-based system, where solution elements may be left in an undetermined state at the end of the process because no rule is applicable in the current circumstances.

Once again, the reasons for these deficiencies might be ascribed to the training data. There are very few examples available, and those that are can appear, to a greater or lesser extent, to be inconsistent, the products of quite different processes and influences – which is hardly surprising, since that is exactly what these design examples are. The inductive machine learning algorithms that are available tend to require large numbers of data, which are assumed to be produced by a consistent system, with known inputs and outputs. This suggests that the blame could equally well be shouldered by the immaturity of these algorithms. Their use here indicates that they are very far from realising general models of human learning processes, which are able to apply background knowledge of the domain and common-sense to assist in learning successfully within such limitations.

This also means that if the knowledge implicit in design examples were to be exploited using available technologies, it would have to be by some other method. Section 3.6 introduced the idea of Case-Based Reasoning, in which examples of previous design episodes are used to suggest the solution to a new design problem. The following chapter presents a variant on the basic CBR model that has been developed to generate conceptual designs of fluid power circuits within the knowledge and data limitations imposed by this particular problem.

11 A Case-Based Reasoning Approach – *Case-Informed Reasoning*

The problems encountered when applying ML algorithms to learning design heuristics suggest that the automatic learning techniques have yet to reach a sufficient level of maturity to be useful for learning about tasks of such complexity.

However, the wish to automate conceptual design synthesis remains, as, unfortunately, does the bottleneck in knowledge acquisition. Despite their limited success, the experiments with ML have not served to disprove the hypothesis that previous design episodes are a useful source of design knowledge. Therefore, the question becomes, does AI offer any other technique by which this knowledge source could be tapped?

The CBR model of design reasoning suggests an answer to this question. In the general form of this model, as introduced in section 3.6, complete design solutions are re-used on the basis of their having successfully met a design specification that is in some way similar to the current problem. In this way, previous design episodes are used explicitly in the problem solving strategy. However, a principal drawback to the CBR approach is that adaptation knowledge is often required to make the retrieved design solution better suited to the meeting the current specification – and this knowledge, usually being of a heuristic nature, is itself susceptible to the bottleneck in acquisition. Furthermore, in this particular case, the archive consists of 30 examples, which provides slight coverage of the number of potential design problems describable using the developed specification representations – and so it is unlikely that any archive solution, used as found, would provide a satisfactory solution to a new problem. This degree of coverage may well be typical of design domains.

To try to overcome this difficulty, a variant of CBR called *Case-Informed Reasoning* (CIR) has been developed in the course of the research described here. The CIR model of the design process is based upon the recall and integration of *elements* of previous design solutions from memory, rather than the recall of a *complete* solution. Each of these elements appears to have solved some aspect of the design problem that is also a feature of the current problem, and, moreover, has done so in an (apparently) similar context (Figure 30). (This is

a form of *constructive CBR*, a general term introduced by Pu (1993) to describe approaches that recall and combine relevant *parts* of past solutions.)

Rather than choosing a complete solution and then adapting it, this approach combines features of solutions to form a new solution that, hopefully, meets the specification. This means that adaptation heuristics are not required to modify the retrieved solution; additional knowledge *is* needed, though, in order to identify the useful elements of solutions. This knowledge is in the form of a *classification* of attributes of the specification and a set of *explanation rules*. As will be seen, these rules are considered to be of a different nature than the sort used in design KBS, as seen in chapters 3 and 8, and this difference would seem to render them more susceptible to acquisition by knowledge engineering techniques.

This chapter is devoted to a description of the CIR approach to the conceptual design of fluid power systems. Before describing in detail the system implementing this approach, the additional knowledge required to implement this approach is introduced.

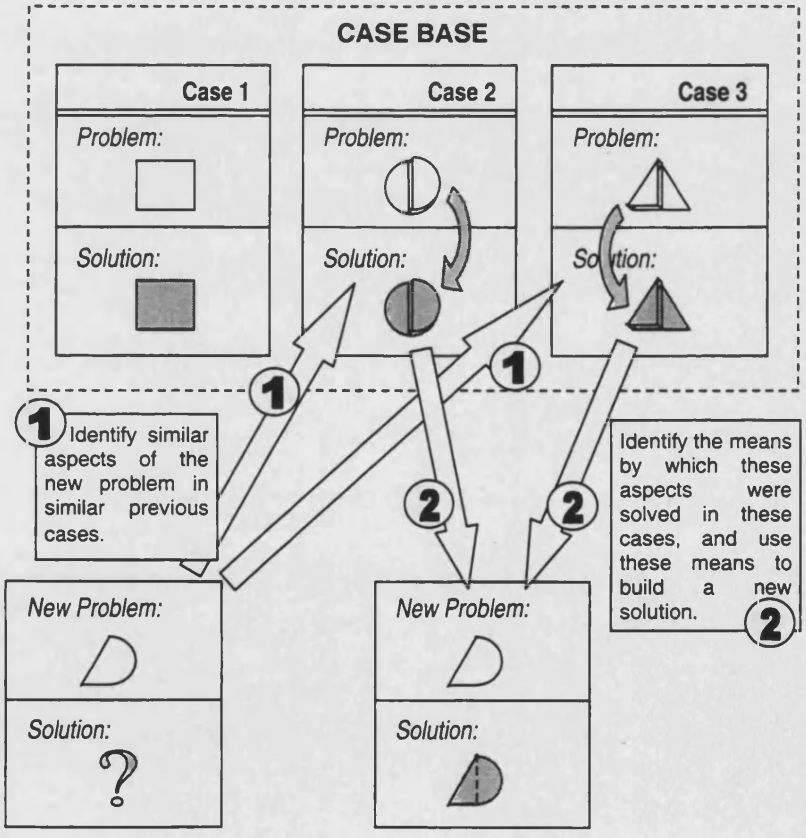


Figure 30. The Case-Informed Reasoning approach; cf. Figure 6, the general CBR approach.

11.1 Additional Knowledge Required for CIR

The CIR approach, then, relies on the introduction of additional knowledge in the forms of a classification of specification attributes and of explanation rules. These both represent additional *domain* knowledge about this task.

11.1.1 Specification Attribute Classification

During the development of the static state specification representation, it was recognised that certain of the attributes seemed to be describing some aspect of the performance or characteristic of the system to be designed as a whole (and so, these are termed *characteristic attributes*). In contrast to these, the remaining attributes refer more directly to the functions that the system must fulfil (*functional attributes*). This distinction is exploited in the CIR model of the design process. Table 19 shows this classification.

attribute name	*type*
maximum load	characteristic
maximum speed	characteristic
plane of motion	characteristic
continuously variable speed	**functional**
hold load stationary	**functional**
smooth accelerations	**functional**
hold load on failure	**functional**
load-independent speed	**functional**
control extend speed	**functional**
control retract speed	**functional**
solution requires motor	**functional**
control inertia	**functional**
energy efficiency paramount	characteristic
control accuracy	characteristic

Table 19. The classification of the design specification attributes.

11.1.2 Explanation Rules

Analytical knowledge of the domain is required in order to decompose the case solutions into useful elements. This knowledge is in the form of *explanation rules*. Each of these rules has the form:

$$\{x_1, x_2, \ldots, x_n\} \Rightarrow f$$

where x_i is some member of the set of solution elements, and f is one of the functional attributes. The rule may be read as, 'this set of solution elements can be used to provide functional attribute f'. There may be a number of different rules for each functional attribute, reflecting the different ways in which the attribute can be achieved. In addition, a particular solution element may be related to more than one functional attribute.

Using these rules, a given design solution may be analysed so as to provide a hypothetical explanation of the manner in which it achieves its functionality (expressed in terms of the functional attributes) – hence the name given to these rules. As described in chapter 5, in the archive of design examples each design solution is paired with the corresponding specification from which it was generated. The rules allow the presence of each element in the solution to be explained as satisfying (or contributing to the satisfaction of) one or more of the demanded functional attributes.

A set of these explanation rules for the conceptual design task as it is represented here has been handcrafted. With all the archive examples expressed appropriately in terms of the static state specification and the solution representations, this was done as follows:

1. Take the next archive example. Associate every element in the solution to this example with one or more of the functional attributes demanded (that is, having the value *yes*) in the specification.

2. For each demanded functional attribute, the set of associated solution elements is used to create a rule of the form given above. If this rule is not already a member of the total set of explanation rules, then it is added to this set.

3. Repeat for the next archive example until every one has been analysed in this way.

In this manner, with the archive containing 30 examples, about 30 different rules have been generated, with each functional attribute referred to by at least one rule. By way of an example, the rules for one particular, relatively simple solution element, POCV_A, are as follows:

{POCV_A} ⇒ hold load stationary

{POCV_A} ⇒ hold load on failure

Under certain conditions, this element stops the flow in the circuit. The rules describe how this behaviour can be used (on its own) to satisfy one of two particular functional attributes (or, potentially, both of them simultaneously). In addition, there are also rules that state:

{CBV1_A&B} ⇒ hold load stationary

{DECV_A&B} ⇒ hold load stationary

{CC_DCV} ⇒ hold load stationary

Given a new specification, in which the functional attribute *hold load stationary* is required, each of these solution elements has the potential to satisfy the function; there is no indication, however, of which should be used to solve this particular problem. Hence, these are not useful design *synthesis* rules: given a particular functional attribute to satisfy, they do not provide enough information to allow the correct choice to be made of one of the sets of solution elements that can be used to achieve that function. What is lacking is some indication of the *context* in which a certain set of elements will achieve the attribute: these are *analytical* rather than *synthetic* rules.

One basis for making this choice lies in the archive of example designs, in which the use of solution elements in context can be seen. With this in mind, a new strategy for generating design solutions can be devised. This, and the design system which implements it, will now be described.

11.2 Intelligent Design System – Case-Informed Reasoning

As for the earlier, ML-based KBSs, the CIR design system will be described according to the knowledge categories introduced in section 3.3. This is followed by a description of the implementation of this model, and of some of the results of testing the system.

11.2.1 Strategic Knowledge

The set of explanation rules and the archive of designs are used to solve new fluid power circuit design problems in the following manner. A new design specification is presented to the system. For each *functional* attribute that is demanded (i.e., has the value *yes*) in this specification, a search is made of the archive to identify those design examples having a specification in which this attribute is also asked for (and, hence, satisfied by the solution). If the attribute is satisfied in more than one example, then the *best* example is that which has a specification that is the most similar *in its entirety* to the new specification. In the event of

several examples being judged equally similar, an arbitrary choice is made amongst them to determine the best (conceivably, these examples could be used to construct alternative solutions).

Once this best example has been found, its design solution is examined, and, using the explanation rules, the set of elements in the solution that provides the functional attribute under consideration is identified. If it does not already exist in the new design solution, each of the elements in this set is then added to it.

If, however, no example of the satisfaction of a particular functional attribute exists in the archive, then an explanation rule associated with that attribute is selected by default to suggest the solution element(s) to use. In such a case, the system has no 'experience' of the attribute being satisfied, so the choice is based on analytical knowledge alone. (However, here there is at least one example of the satisfaction of each functional attribute in the archive.)

The process is then repeated for the next functional attribute demanded, and so on, until all have been satisfied, at which point, a complete design solution is considered to have been constructed. Thus, the process is one of finding solution elements which, according to the explanation rules, provide some functionality that is required to solve the current design problem, and moreover, which are recalled as having been used to do so in a context which is similar to that of the current problem.

To summarise, the design algorithm is as follows (Figure 31):

1. Let D, the set of current solution elements, be empty. Let S be the (user-supplied) specification of the new design problem.
2. Take the next functional attribute, f, that is demanded in S. If all have been considered, then stop: D is the suggested design solution to S.
3. Compare S with the specification of each example in the archive, and denote as *best* that example having the specification most similar to S, and in which f has been satisfied. If all the examples have been examined and no *best* has been found, go to stage 5.
4. Using the explanation rules for f, examine the solution of *best* to determine which solution element(s) satisfy f in this case. In the case of more than one rule being applicable, that referring to the largest set of elements is used. Add these element(s) to D and return to stage 2, or, if no explanation is possible using the rules, go to stage 5.
5. Using the explanation rules for f, add a default solution element(s) to D. Continue from stage 2.

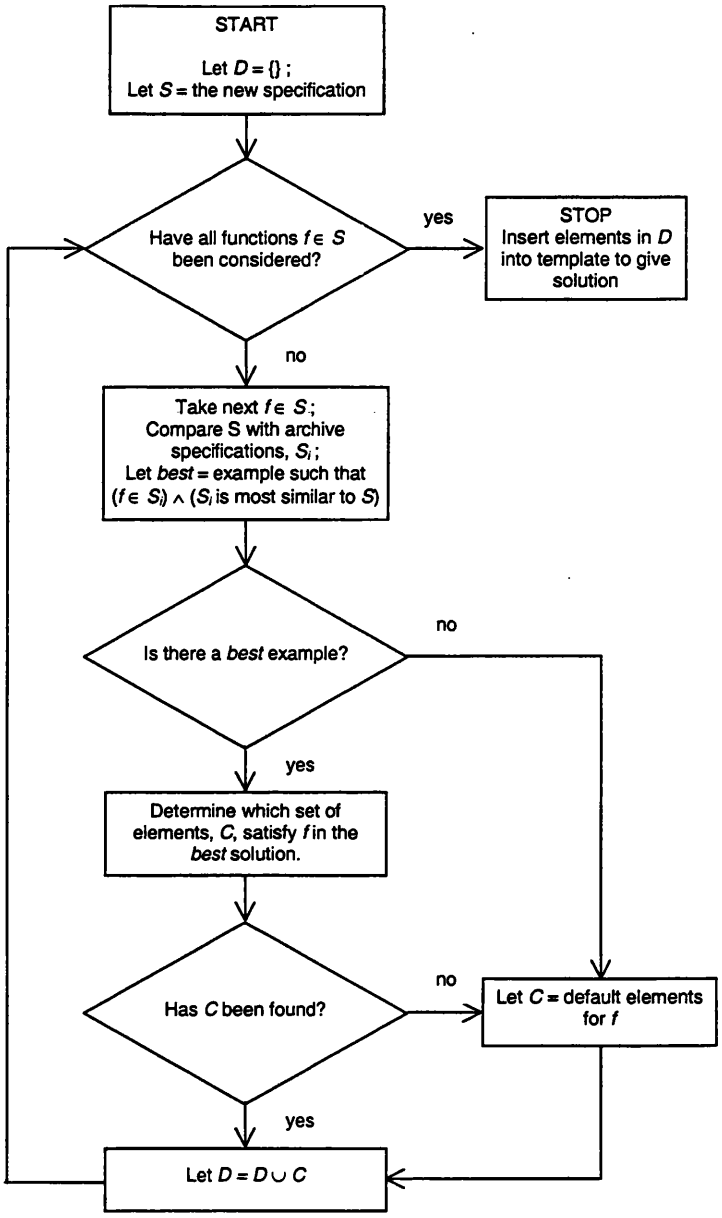


Figure 31. The CIR algorithm.

11.2.2 Inference Knowledge

The inference knowledge in this model takes the form of the knowledge that is used to retrieve the best matching case and extract the relevant solution elements for use in the new solution.

11.2.3 Domain Knowledge

The domain knowledge is composed of:

- the specification attributes (static state representation), their values and their classification as either characteristic or functional;
- the order in which to consider functional attributes during the application of the strategy (here, though, the functional attributes are considered in an arbitrary order);
- the solution elements;
- the set of explanation rules, and;
- the archive of previous successful design examples.

11.2.4 Working Knowledge

The working knowledge comprises the current specification, best matching examples, and the functional attributes that have been satisfied and the solution elements selected to do this.

11.2.5 Implementation

The facet of the inference knowledge that remains vague in this model is the method by which the similarity of specifications is determined. Here, for this purpose, a simple Hamming distance-type measure is applied to determine the 'distance' of the specification of each archive example from the new specification. The number of values of corresponding attributes (of both types) that differ between the two specifications is the distance between them. That example with a specification having the fewest differences from the new specification is considered to be the best matching case.

Table 20 gives an example of this matching process for one of the functional attributes, with an archive of 4 examples. Here, considering the functional attribute *hold load stationary*, asked for in the new specification, two of the examples, numbers 1 and 3, have solutions satisfying this attribute. Of these, example 3 is the judged to be the closest using the distance measure – its specification differs from the new one at only 4 points, whereas there are 6 differences from the specification of example 1. Therefore, the solution element(s) used to *hold load stationary* in example 3 will be those used in the new solution. In examining the solution to example 3, the solution element *POCV_A* may be found. From the explanation rules, it is known that one of the uses of this element is to provide this required functional attribute, and so this is the element that will be used in the new solution for this purpose.

Attribute	*Type*	*New*	*Example 1*	*Example 2*	*Example 3*	*Example 4*
maximum load/force	characteristic	high	high	low	low	high
maximum speed	characteristic	low	high	high	low	low
plane of motion	characteristic	horiz.	non-horiz.	horiz.	horiz.	non-horiz.
continuously variable speed	functional	no	yes	no	yes	yes
hold load stationary	functional	yes	yes	no	yes	no
smooth accelerations	functional	no	no	no	yes	yes
hold load on failure	functional	yes	yes	yes	no	no
load-independent speed	functional	no	yes	yes	no	yes
control extend speed	functional	yes	yes	yes	yes	yes
control retract speed	functional	no	no	yes	no	yes
solution requires motor	functional	no	yes	no	no	yes
control inertia	functional	yes	yes	no	yes	yes
energy efficiency paramount	characteristic	no	no	yes	no	no
control accuracy	characteristic	high	low	high	high	low
'Distance' from new specification:			6	7	4	9

Table 20. An illustration of the adopted matching metric with an archive of 4 examples.

This is an unsophisticated measure, and takes advantage of the somewhat simplistic manner in which the specifications are represented here. Better solutions would follow from using a more sophisticated matching algorithm, which sets greater store upon matches with attributes influential in the satisfaction of the current function, while disregarding those irrelevant to the selection of elements. However, as mentioned in section 3.6, the implementation of this would require additional (rather complex) domain knowledge to be acquired and added to the system. For this reason, this simple, domain-independent metric was used here.

In the event of there being two or more archive examples that include the satisfaction of the current functional attribute and which have specifications found to be equally similar to the new specification, in the implementation described here an arbitrary choice is made as to which is to be used in forming the new solution. This decision might profitably be made on some other basis. For instance, the description of the archive given in appendix A includes an indication of the degree of confidence in the correctness of the solution to each archive example; that example having a solution in which there is the greatest confidence might be that selected for use.

11.2.6 Results

The CIR model described above has been implemented using an archive of 30 examples. As would be expected, given the use of the explanation rules in the algorithm, this system provides solutions in which all the desired functionality is embodied in some identifiable form. To this extent, all the solutions that are generated *appear* to be plausible. This also has the effect of making redundant the testing method described above in section 6.3.1 that was

adopted for the ML-based approaches, since all the solutions appear to fall into categories 1 and 2. (For the record, 13 test cases produced category 1 solutions, the remaining 7 falling into category 2, indicating some degree of over-engineering.)

To illustrate the operation of the system, the test case shown in the second column of Table 21 will be used (this is the same example test case used in sections 7.4.6, 7.5.6 and 8.2.6). Now, there are four functional attributes demanded in this specification, namely, *continuously variable speed*, *smooth accelerations*, *solution requires motor* and *control inertia*. In the archive, there happens to be an example (archive number 11) with the specification shown in the third column of the table. It will be seen that all of the functional attributes desired of the new solution are satisfied in this example, and that the specifications are quite similar (a distance of 3 separates them). This is sufficient to provide a match for all these functions, since no other relevant specification is closer to the test case.

attribute name	*Test value*	*Archive No. 11*
maximum load/force	high	high
maximum speed	low	medium
plane of motion	horizontal	horizontal
continuously variable speed	yes	yes
hold load stationary	no	yes
smooth accelerations	yes	yes
hold load on failure	no	yes
load-independent speed	no	no
control extend speed	no	no
control retract speed	no	no
solution requires motor	yes	yes
control inertia	yes	yes
energy efficiency paramount	no	no
control accuracy	high	high

Table 21. Test case specification.

So, considering each of the functional attributes in turn:

- *continuously variable speed* – so, archive case 11 is the closest match satisfying this attribute. The solution to this case is examined. The solution contains the element *PROP_DCV*, and, from the explanation rule:

  {PROP_DCV} ⇒ continuously variable speed

  it is known that this element provides the functionality, and so it is added to the solution.

- *smooth accelerations* – with archive case 11 again providing the match, the presence of the element *CBV_A&B* in the solution, and the known rule:

  {CBV1_A&B} ⇒ smooth accelerations

  dictate that this element is added to the solution.

- *solution requires motor* – as above, but a more trivial inference in this instance, case 11 includes a motor, which is added to the solution.

- *control inertia* – again, the solution to case 11 is examined, and the rule:

  {CBV1_A&B, CVC_A&B, PRV2_A&B} ⇒ control inertia

  is used to add the elements *CVC_A&B* and *PRV2_A&B* to the solution (*CBV1_A&B* already having been added to provide *smooth accelerations*). For this particular function, the explanation rules also state:

  {CVC_A&B, PRV2_A&B} ⇒ control inertia

  {CVC_A&B} ⇒ control inertia

  {PRV2_A&B} ⇒ control inertia

  In other words, subsets of the elements from the first rule can be used to satisfy the function. It was necessary to search the entire solution to find the largest set of elements that provide *control inertia* in order to ensure that it is provided in the new solution. In archive case 11, it may be that the *high* load magnitude means that all three elements are needed to achieve the function (with *CBV1_A&B* also providing the necessary *smooth accelerations*). If this is the case, then their use, under similar load characteristics, seems appropriate in the new solution.

So, then, a new solution has been constructed (Figure 32), in which all the functionality is satisfied, and which does not appear to have any superfluous elements – this is a category 1 solution. There is a degree of confidence in this solution provided by the fact that an example – archive case 11 – has, in similar circumstances, successfully provided this functionality. It is important to note that it is not necessarily the case that all the elements of the new solution originate from the same archive example, as here - in fact it is quite unlikely to be the case. If, for instance, archive case 11 did not satisfy *continuously variable*

speed, it would be necessary to look to a different archive case for its solution, an archive case that will, in all probability, be more 'remote' from the current problem.

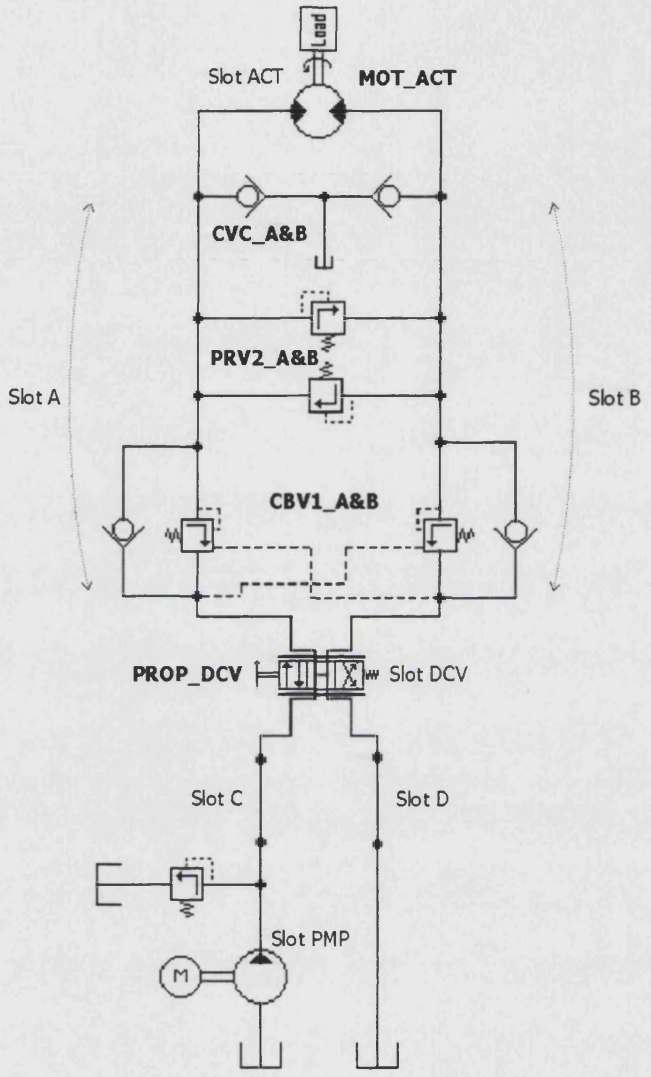


Figure 32. A circuit solution generated by the CIR system.

With the near-redundancy of the testing method, a more important question, then, is that of the extent to which the generated solutions embody design *expertise*, rather than mere competence.[8] In other words, whether, in response to the current specification, the combinations of solution elements correspond to sensible, 'good' configurations of the sort that a human expert might suggest. The degree to which expertise is embodied in a solution

[8] The inadequacy of the testing method is reinforced by the fact that the classification rule-based KBS, when presented with the same test case, also produced a category 1 solution (section 8.2.6). However, this is a different solution from the one generated by the CIR system – which, if either, is the better? Why?

is extremely difficult to gauge: whereas the previous testing could be done by a human having the sort of analytical knowledge expressed in the explanation rules, assessing the expertise of a solution would seem to require either a human design expert able to make this judgement, or else the physical construction and testing of the circuit. Since the former is unavailable, and the latter impractical, the expertise imparted by this approach remains an open question. (More generally, as mentioned in the preceding chapter, this sort of assessment of any automatic design system is difficult, and there is little discussion in the literature on the subject.)

11.3 The CIR Approach - Discussion

The CIR approach to design synthesis relies on the assumption that, since they have been used successfully in similar situations in the past, the retrieved solution elements will be applicable to the current problem – this assumption is unlikely to hold in every case.

Furthermore, the method relies on a number of features of the fluid power design task that are not necessarily shared by all conceptual engineering design tasks. In particular, the design specification can be stated in such a way that its expression of the desired functionality can be directly related to elements or sets of elements within designs. In other design tasks, the functionality may be much more a consequence of the global system, in which case the explanation rules cannot exist in the form that they have here, and so their use to decompose solutions according to local functionality will not be possible. In addition, a rather restricted description of the specification has been adopted, which simplifies both the rules and the matching algorithm used. However, the expression of such rules *does* seem valid in this domain.

Obviously, there is much potential for improving upon the methodology as it has been implemented. As mentioned above, the matching algorithm is unsophisticated (and as such, would seem to undermine some of the claims of expertise for the system). Furthermore, the strategy involves examining and satisfying functional attributes in a sequential fashion, which would seem to make the process subject to problems of over-engineering, since a solution element can often achieve more than one of the required functions. However, the element might not be able to do so in the current problem context, so a simple check, using the explanation rules, of the 'functionality' already supplied by a partial solution would be inadequate. An additional point raised is that of the order in which the functional attributes are considered and satisfied, since each is considered successively. Some orderings may allow more optimal solutions to be derived. For example, by satisfying more complex

attributes initially, the less complex ones may be already satisfied adequately. Again, this would require additional domain knowledge to be incorporated within the algorithm.

11.3.1 The Acquisition of the Explanation Rules

The problem remains, though, of acquiring the explanation rules necessary for this approach. For the implementation described here, the rules have been handcrafted. Since this has been done by someone with limited experience of the domain, there is the strong possibility that some of these rules are not wholly accurate. However the analytical nature of the rules and their relatively low complexity (even when compared with, say, the sort of rules generated by CN2 seen in section 8.2.5) would seem to make them amenable to capture through a knowledge engineering approach of interrogating a human who has a greater familiarity with the domain.

This acquisition process might proceed in a manner similar to that outlined above in section 11.1.2. If the human expert were unable to suggest an association between requested functional attributes and the elements of the corresponding solution, this might indicate a problem with the representations of this design task or with the archive example in question. In either case, this may provide a context for the expert to suggest how the problem should be rectified.

A preliminary investigation into this means of knowledge acquisition with a design expert suggested that, properly presented and structured, this could indeed be a more reasonable and tractable method than is asking experts for their synthesis or modification heuristics. However, the unavailability of an expert for the length of time that would be necessary has meant that no methodical study of the process has been attempted.

11.3.2 Machine Learning the Explanation Rules

Given the difficulties encountered when applying ML algorithms to learning design synthesis heuristics, the use of the simpler analytical knowledge in the CIR model raises the question, could ML be used to acquire the explanation rules? If so, an archive of examples could be used to generate the rules, and then could be used as the experiential basis for choosing which elements to include in a solution. If it were possible to do this, it would effectively remove the need to perform any knowledge engineering whatsoever for the acquisition of these rules.

A somewhat similar approach has been suggested by Thompson and Mooney (1994) for their medical diagnosis abduction system. They induce an 'abductive knowledge base' of rules of the form:

$$\text{disorder} \Rightarrow \text{symptom}$$

where '$\Rightarrow$' is read as 'causes', which corresponds to the explanation rules used here, where solution elements are the 'disorders' that cause specification attribute 'symptoms'. They discuss the application of their own *Learning for Abduction* (LAB) algorithm (employing a variant on associative learning - see section 4.4) to the task of acquiring these rules from 50 example diagnoses (consisting of sets of disorders and the corresponding symptom). It is found to be able to construct successfully a knowledge base of rules.

APRIORI and Explanation Rules

A similar approach was tried here, using the APRIORI associative learning algorithm (section 4.4.1). However, with 17 solution element 'disorders' and only 30 examples, a typical element is *present* in only a handful of examples. As a consequence of this, in general, an element will be associated not only with the functional attributes that it satisfies, but also with the functional attributes that, by coincidence, are present in every solution in which the element is also present. The result of this is that, while all of the explanation rules actually used in the implementation are discovered in this fashion, a good many spurious rules are also found, and it is impossible to distinguish between the good and the spurious. (Each of Thompson and Mooney's examples has, on average, less than 2 disorders (fewer than is the case here), and so their set of 50 examples seems sufficient to produce a useful rule set. Furthermore, their domain is simpler in that diagnoses are assumed always to explain their associated symptoms. This is not necessarily the case in design. For example, a certain solution element may *control inertia* under particular (load, speed, etc) conditions, but not in others. So, the presence of other features in the design 'symptoms' can influence the relationship that holds between cause and effect.)

CN2 and Explanation Rules

The CN2 algorithm was also applied, to try to learn the rules for classifying whether a functional attribute is *present* or *absent*, based on the elements present in a solution. The problem here is that too *few* rules are generated – the algorithm tends to generalise too much, and so omits rules. This is another effect of the complexity of the domain. If the data contains an example in which a certain combination of elements is required to satisfy a function and also contains a second example in which (under different conditions) some

subset of this combination is used to achieve the same function, then the algorithm generalises to suggest that the subset *alone* is sufficient to fulfil the function. Hence, in such circumstances, only one explanation rule is found, with one or more valid explanation rules missed.

These problems meant that the attempt to machine learn the rules was abandoned. It remains, though, an intriguing proposition. Learning the *analytical* rules does not seem to be so difficult a task as is learning *synthesis* rules. With a more sophisticated learning algorithm or more examples, the rules could be extracted from an archive. With these rules, the CIR system could then synthesise solutions, using different, contextual, information from the same archive.

11.4 Learning in the CIR System

This design system has an additional advantage over most ML-based systems, in that it is easier to introduce some form of incremental learning, whereby the system can gradually be improved during its lifetime, and without the need for extensive re-implementation. A new solution (which may or may not have been generated by the system itself) that is found or judged (by whatever means) to be a good one, can be added to the archive, by encoding its specification and solution appropriately. This solution can then be used in subsequent design episodes to influence the solution to problems having a similar specification. In this way, new design experiences can be incorporated and used when they become available – there is no need for retraining, as there would be with the non-incremental ML approaches.

However, difficulties arise with this approach if the new example does not conform with the existing ones. It may be the case, for instance, that the specification or solution representations are inadequate to describe the example, or that solution elements are used in a manner that is not described by the set of explanation rules. To incorporate such an example, it may be necessary to modify the representations (which would mean that all existing examples in the archive would have to be re-described), or add further rules to explain its functionality.

This approach also provides a means by which the behaviour of the system can be corrected. If a solution that is generated by the system is found to be a poor one, the introduction into the archive of a better solution (generated by some external agent) to that particular specification would prevent the same failure occurring in the future and, hopefully, improve the system's response to similar specifications.

11.5 CIR as Abductive Reasoning

Leake (1993) has noted that CBR provides a model of abductive explanation generation. He says:

> *"The case-based process...facilitates generation of plausible and useful explanations despite the problems of incomplete information and imperfect domain theories that mark everyday explanation."*

Here, the domain theory is obviously lacking – the heuristics that govern the synthesis of circuits are not available – and the specification representation is not (and perhaps cannot) contain *all* the information relevant to producing design solutions. Leake goes on to say:

> *"...the case-based approach helps to choose between competing explanations licensed by an inconsistent domain theory. By favoring explanations supported by specific similar experiences, case-based explanation takes advantage of regularities in the world – similar events are explained in similar ways – even if those regularities are not fully captured by the explainer's domain theory."*

So, in design contexts, the CBR model uses its 'experience' to form solution explanations of a new design specification. This experience is in the form of a case-base of previous design examples. Having made the assumption that similar specifications are satisfied by similar solutions, the example having the most similar specification is found, and its solution circuit used to generate a hypothesis as to how this specification may be brought about.

As such, this can be seen as a 'subjective' method – solutions are generated based on the current contents of the system's case 'memory' – and it may not be possible to generate a wholly satisfactory solution using the knowledge that the system currently has. In this, CBR would seem to emulate some of the qualities of human abductive reasoning, where, faced with a lack of certain knowledge, an appeal is made to experience to help solve the current problem.

In the CIR model described above, the solution element 'causes' of particular functional attribute 'effects' are stated explicitly by the explanation rules. These rules provide some additional domain theory for this particular design task. Since there can be more than one rule for each functional attribute, it can have more than one cause. What these rules lack, however, is the information that would allow one cause to be preferred over the others, that would allow a hypothesis to be proposed about how to satisfy the function for the current problem.

The information that will permit the choice of appropriate cause lies in the archive of examples. Each example in the archive provides experience of the successful satisfaction of a number of functional attributes. The system uses the most similar experience of the function being supplied as the basis (and justification) for the choice among the competing solution element causes. The sum of these hypotheses for all the demanded functions gives the proposed solution.

In chapter 6, the adopted ML-based KBS approach to conceptual design synthesis allowed some answers to be suggested to the questions concerning the manner in which successful abductive inferences are made. The same can now be done for the CIR approach. To recap, these questions, first introduced in chapter 2, are as follows:

- what triggers abductive reasoning?
- what mechanism controls the generation of abductive explanations?
- upon what criteria is the judgement of the 'best' explanation based?

11.5.1 Triggering Abduction

As for the ML approaches, this abductive episode is triggered by the receipt of a new, complete design specification, described according to the devised representation. An acceptable cause is a complete circuit solution, described in terms of the template and a set of solution elements, in which each of the required functional attributes is satisfied.

11.5.2 Abductive Reasoning Mechanism

More formally, for each function f_i that is demanded in the specification, a search is made of the available archive examples to find that example having the most similar specification that also includes this function. From the solution, it is found that the satisfaction of this function is explained through the use of the rule:

$$\{x_1, x_2, \ldots, x_n\} \Rightarrow f_i$$

where x_j is some member of the total set of solution elements. (To reiterate, this rule may be read as, 'this set of solution elements can be used to provide functional attribute f_i'.) Now, let S_B be the specification set of this best-matching example, and S_N be the specification set for the new design problem. Assuming that the solution is correct (and that the representations are complete and accurate models of the domain knowledge), it can be asserted that:

$$S_B \cup \{x_1, x_2, \ldots, x_n\} \rightarrow f_i$$

or, in other words, that this set of elements, in this specification context logically implies (the satisfaction of) this function – in this context, these elements provide this function. Relying on the judgement of S_B and S_N being the most similar (according to available experience), an assertion can then be made to the effect that:

$$S_N \cup \{x_1, x_2, \ldots, x_n\} \rightarrow f_i$$

or, in other words, in the new context, this same set of elements will achieve the same function. This is, in effect, the rule to be used abductively in this case. 'Inverting' the function and solution elements gives a design heuristic for this design episode:

$$S_N \cup f_i \rightarrow \{x_1, x_2, \ldots, x_n\}$$

or, since S_N necessarily includes f_i, more simply:

$$S_N \rightarrow \{x_1, x_2, \ldots, x_n\}$$

This synthesis heuristic can now be used in a process of 'probable deduction' as follows:

$S_N \rightarrow \{x_1, x_2, \ldots, x_n\}$	*rule*
S_N	*premise (the given specification)*
———	
$\{x_1, x_2, \ldots, x_n\}$	*conclusion* $(= C)$

and so this set of elements, C, is postulated as the means to achieve the function in this case. With m functional attributes asked for in the specification, similar arguments allow m conclusion sets of elements to be inferred:

$S_N \rightarrow C_1,\ S_N \rightarrow C_2, \ldots, S_N \rightarrow C_m$	*rules*
S_N	*premise*
———	
$C_1,\ C_2, \ldots, C_m$	*conclusions*

and a further assertion is made to the effect that the union of the m sets of solution elements provides the design solution to the new problem:

$$\textit{solution, } D = \bigcup_m C_i$$

The strategic knowledge for this model of the design process, then, resides in this mechanism of matching specifications and manipulating explanation rules into synthetic

heuristic forms, along with the various assumptions made along the way. These heuristics are, on the whole, logically unsound, but may be justified by their *usefulness* in generating a solution. They embody the abductive heuristics for making the choice of *most likely* design solution 'cause' of the specification 'effect'.

11.5.3 Making the 'Best' Decisions

In this abductive reasoning mechanism, the notion of the 'best' or 'most likely' hypothesis is implicit in the idea that similar problems are solved in similar ways, and so, the best hypothesis is that which has been seen to be successful in a similar situation in the past. So, the choice of the most likely set of solution elements to use for each function is dependent on the particular similarity measure that has been implemented. In this case, it is a simple measure of the 'distance' between the values of the new and the existing specifications. While this is a crude measure, in the absence of a more advanced metric, it provides a basis for inferring which solution element is the most likely to be applicable in the current context. The CIR methodology implies that each of these independent 'best solution element hypotheses' will, when put together, form the best complete solution hypothesis. This assumption is not necessarily a sound one; however, it would require further domain knowledge to identify those situations when incompatible elements impair the quality of the whole circuit. Once again, this knowledge is not readily available.

The method allows generalisation, of a certain form, over the domain. In the model, an implicit assertion is made to the effect that the solution elements used to fulfil a certain function in a context most similar to the current problem will be the most appropriate way of satisfying it in the current solution. Assuming that there is at least one example in the archive of the satisfaction of each functional attribute, there will always be a 'most similar' context – although, of course, with a limited archive the 'most similar' context may, in fact, be quite different. (This also means that the addition of further examples to the archive should have the overall effect of improving system performance, always assuming that the measure of similarity is a useful one.) In this manner, some response can be made to every new design specification. The degree of confidence in the appropriateness of the selected elements is directly related to the degree of similarity of the best matching example. The distance measure, then, may be seen as providing a measure of the confidence in each of the choices.

In the event of no instance of the satisfaction of a particular function being available in the archive, some degree of completeness in the abductive reasoning could be provided through the selection of a default set of solution elements. In such circumstances, the choice is no

lon
ele
ratl

longer based on direct experience. However, the criteria for marking a particular set of elements as the default should perhaps be based on the set's simplicity or cheapness, say, rather than being arbitrary choice.

To
this
lea
the

To conclude the discussion of the abductive nature of this model, the distinction between this approach and the previous ML approaches should be noted. In the latter, the goal was to learn the abductive *heuristics* that can then be used to perform this design task. In contrast, the CIR approach abductively 'learns' *a solution* to the current problem.

11

11.6 Building a CIR System in Other Domains

The
tasl

The sequence of steps by which a CIR system might be constructed for conceptual design tasks in other domains can be stated in general terms.

1.

1. The first step involves understanding the design task within the chosen domain, and then generating representations of the design specifications and solutions, in a manner similar to that described in chapter 5. Within the specification representation, those attributes that refer directly to aspects of the desired functionality should be identified. The solution representation should be in terms of the functional elements that are used as the 'building blocks' of systems in the domain.

2.

2. Examples of the design task must then be collected. Each example should consist of a specification and the corresponding solution and be described according to the developed representations.

3.

3. For each of the design examples, some explanation of the manner in which each of the functional attributes is achieved by elements of the solution must be proposed. Using this information, a set of explanation rules can be generated in a manner similar to that outlined in section 11.1.2.

4.

4. Some metric for judging the similarity of two design specifications must be generated. In this case, the similarity metric used is domain-independent, but, if the appropriate knowledge is available in other domains, it might usefully incorporate a deeper understanding of the task.

5.

5. An appropriate order in which to consider the functional attributes of the specification when generating a new solution must be proposed.

6. Using these constituents, the algorithm shown in Figure 31 can be implemented. This implementation represents a CIR system for the design task.

A failure to satisfactorily complete any of these steps might indicate that the required knowledge is not available or accessible, or else that the design task in hand is not suited to the CIR approach. In addition, an implemented system that is found to perform poorly might suggest that inappropriate choices have been made for one or more of the constituents of the system.

11.7 CIR Approach – Conclusions

The CIR approach described here is an attempt to harness the design expertise implicit in a set of design examples. A new design problem is presented in terms of the functionality required of a solution. The examples are searched to determine which solution elements have been used to provide this functionality in similar contexts in the past. These elements are then used to construct a circuit solution to the new problem. This approach is an attempt to exploit a relatively small set of examples by recombining parts of them to solve new problems. It relies on system functionality being embodied locally in sub-sets of solution elements (and the relationships between functions and solution elements being known).

In this model, unlike the ML approaches, no attempt is made to integrate the expertise implicit in the examples into a coherent body of compiled synthesis heuristics. Accordingly, this technique is less vulnerable to the problems of inconsistencies in the data, whereby differences in style, content and type of design can prevent the archive from appearing to present a coherent description of this design problem, which, as a result, can hinder learning. (In fact, the CIR approach can accommodate wholly contradictory examples – two examples having the same specifications but different solutions – by making an arbitrary choice of which of the two to use when needed during the process.) The design examples are stored in the system 'memory' explicitly, with the expertise they contain remaining implicit until it is needed to solve a problem. At that time, a synthesis heuristic, expressing how to satisfy a function in a particular context, is developed using a specific example and re-applied to satisfy the function in the new solution.

In addition, some indication of the confidence in the choices of solution elements can be gained from the 'distance' between the current specification and that of the example judged to be the most similar. A shorter distance means that the solution element has been used to solve a problem that is more similar to the current problem, and so, is more likely to be applicable in the current context. This indication of the confidence in the rationale for each

design decision would seem to be a very useful feature of any automated approach to design, and one which is singularly lacking in the ANN-based approaches (but which is present in the classification knowledge produced by the CN2 algorithm).

It should be noted, however, that the CIR system would seem to have an immediate advantage over the ML-based approaches. It has access to additional domain knowledge, in the form of the explanation rules and the classification of the specification attributes, knowledge which is denied to the earlier approaches. The explanation rules are analytical in nature: they permit a circuit solution to be deconstructed into its functional elements. As such, they are not synthesis heuristics – on their own, they do not allow a solution to be constructed in any rational manner – but, in combination with the archive, they do allow synthesis to occur. Their analytical nature would also seem to render them more susceptible to capture through conventional knowledge engineering techniques (and, potentially, through machine learning).

Many of the details of the algorithm could be altered to try to improve performance. However, in a sense, these details are immaterial - the essential concept here is the use of a set of design examples, plus analytical knowledge of the domain, to synthesise design scheme solutions to new design problems. With a limited case base, conventional CBR approaches require secondary solution modification heuristics, which are difficult to acquire. Here, the emphasis lies in constructing the right solution in the first place, using only those solution elements that are needed to solve the current design problem.

This model can be seen as implementing abductive reasoning, with the design experience in the archive providing the basis for selecting the 'most likely' solution hypothesis. In this way, the model implements the sort of reasoning that is necessary for generating design solutions. Given the problems associated with the acquisition of the inference heuristics through knowledge engineering and (as seen here) machine learning approaches, the CIR approach offers a method by which the knowledge that *is* available, in the form of the archive and the explanation rules, can be used to propose novel conceptual designs.

12 Conclusions and Future Research

The conceptual design phase of the mechanical engineering design task involves establishing the physical means by which the essential functionality required of a solution is to be achieved. The eventual success, or otherwise, of the resulting artefact or system is often determined during this phase. Conceptual design can be a difficult and complex task, and one that places great demands on the capabilities of designers. The accumulation of the experience necessary for performing this task can take many years – as a result, design expertise is a valuable commodity. The automation of this phase would bring a number of benefits: in particular, it would enable the preservation of valuable design knowledge within an organisation and provide the ability to duplicate and transmit this knowledge throughout the organisation.

The essential element of conceptual design is the *synthesis* of a solution. In complex domains, no algorithms or methods that can guarantee successful synthesis are available – instead, the process is reliant on the creativity and experience of the designer. For the automation of processes for which an explicit algorithm is lacking, conventional computer programs are of little use; instead, it seems necessary to make use of artificial intelligence techniques. AI research is expressly concerned with the emulation of intelligent, human behaviour, such as that seen during the conceptual design activity.

Without an explicit method for conceptual design, expert designers seem to employ *heuristic* knowledge, 'rules of thumb' derived from experience, which, while never being able to ensure a correct solution are, nonetheless, useful, and perhaps necessary, for developing a design. Previous attempts at automating design synthesis can be classified into one of two principal approaches – the *Knowledge-Based Systems* approach and the *Case-Based Reasoning* approach. In KBSs, the heuristics are explicitly expressed in the form of, for instance, design rules, whereas in CBR systems, some or all of the heuristics remain implicit in examples of previous problem-solving episodes. In the construction of design KBSs, the most common strategy has been the use of *knowledge engineering* techniques to acquire the necessary heuristics from human experts. However, this approach has been found to produce

knowledge of dubious quality – experts seem rarely able to articulate completely and accurately their heuristic knowledge.

The use of examples, rather than explicit design heuristics, in CBR might appear to offer a way of avoiding this problem. In practice, however, it would seem likely that the number of available examples of complex design episodes will often be far fewer than the number of potential problems, and so recourse is made to secondary, heuristic knowledge to modify the retrieved solutions to better solve the problem.

Nonetheless, the idea that previous examples are a potential source of conceptual design expertise suggests the research hypothesis adopted here:

> *Artificial Intelligence techniques can be used to access the design synthesis knowledge implicit in previous designs and then re-apply it to produce conceptual solutions to new design problems.*

More specifically, in the context of the research reported here, it is asserted that examples of the translation of design specifications into design solutions contain implicitly the heuristic knowledge that was used to bring about this translation. The research, then, has been concerned with methods of accessing this knowledge and re-using it during new design episodes – and, in so doing, constructing intelligent design systems without incurring the problems associated with knowledge engineering.

12.1 Conclusions

The chosen application domain is that of fluid power systems. Conceptual design in this domain is a configuration design task, involving the selection and combination of a number of standard domain elements into a system that meets the design specification. The findings of this research can be divided into a number of areas. These will now be discussed.

12.1.1 The Logic of Conceptual Design Synthesis

A number of researchers in the field of design have maintained that design synthesis involves *abductive* reasoning on the part of the designer. In other words, the designer proposes a design solution as the 'cause' of the specification 'effect'. However, it has been pointed out that this cannot represent an accurate, general model of conceptual design, since it supposes that the solution, and its relationship with the specification, is already known to the designer, when the design task is precisely one of developing this solution.

Here, however, the task is a configuration design one – so, while the complete solution may not be known to the designer beforehand, the 'building blocks' of this solution *are* known. This allows the task to be thought of in terms of a series of abductive processes, as in the treatment given in the second chapter of this thesis. This treatment provides a model for the heuristics necessary for this sort of design task; however they are expressed, they should relate elements of the specification in particular contexts to elements of the solution, which will then allow a solution to the design problem to be constructed through a process of 'probable deduction'.

12.1.2 The Representation of the Design Problem

A preliminary task is to develop some manner of representing this problem in a tractable fashion that would allow both design specifications and design solutions to be manipulated by computer. To constrain the task within manageable bounds, the decision was taken to focus on the design of single pump-single actuator fluid power *circuits*. As considered here, the conceptual design of these circuits is concerned predominantly with ensuring that the developed solution will provide the necessary functionality, rather than being overly concerned at this stage with questions of spatial constraints, performance levels and so on. Accordingly the representations of the specification and solution that have been developed are focused upon the expression of this functionality.

Two distinct representations of specifications have been developed, the *temporal state* representation and the *static state* representation. The former describes the functionality at the point of actuation in terms of a mixture of quantitative and qualitative attributes over a number of sequential periods in time. The latter representation describes the functionality of the circuit over the whole of its operating cycle in terms a set of qualitative attributes.

A representation of circuit solutions has been devised that makes use of a circuit *template* common to the majority of solutions. This template provides the basic functionality expected of all circuits, with members of a set of *solution elements* providing the additional functionality demanded in the specification.

The development of these representations is not a simple process. Here, it was achieved through a process of *handcrafting*, that is, familiarisation with the domain and the task on the part of the intelligent system developer. A poor choice of representations can render subsequent attempts to reason using them all but impossible, whereas a propitious choice can facilitate substantially this reasoning. The representations can easily be influenced by the desire to use certain computational techniques. A wholly successful automated design

system would indicate that the choice of representations was an appropriate one; a poor design system, on the other hand, indicates little about the quality of its representations, since there may be a number of other factors contributing to its unsatisfactory performance.

Hence, it is difficult to say with any certainty that the representations presented here are appropriate for this design task. However, in the case of the specification representations it may be said that they do not seem to be particularly natural as *human* representations of the task, which is a limitation, since the intention is to emulate the process as a human might address it. The representations are in the form of restricted sets of attributes, and for the static state representation the problem is exacerbated by restricting the range of values that these attributes can assume to, for the most part, binary sets.

For the most part, these representational decisions have been enforced by the desire to use the previous design examples as a source of design heuristics. As seen in this thesis, the AI technologies currently available suggest two approaches to exploiting this source – inductive machine learning and case-based reasoning. However, for either approach to succeed given their current levels of sophistication, it is essential that the examples together present a consistent description of the problem, and that, as far as is possible, regularities in the data are emphasised. This often entails describing the problem in terms of fixed sets of attributes, each drawing from a limited set of possible values – which is the case here.[9] (The decision to move from the more expressive and, arguably, more natural temporal state specification to the static state representation was made expressly to try to accentuate the regularities in the available examples.)

12.1.3 The Archive of Examples of Previous Designs

It has been necessary to develop an archive of examples of previous, successful fluid power circuit design episodes. Each of these examples consists of a design specification and the corresponding design solution. It was found in the course of this development that available examples are relatively few in number, that they are often incomplete and so require varying degrees of reverse-engineering to render them useful and that they are frequently inconsistent, due to the different experiences and capabilities of, and resources available to, their designers. The approaches developed to exploit this archive would have to operate

[9] It might be argued that human designers internally translate the looser specifications they receive into descriptions of this restricted sort, in order to recognise similarities with previous design experiences. However, if this were so, the acquisition of these 'processed' specifications would presumably require a great deal of knowledge engineering, if it were possible at all.

within these limitations. These 'difficulties' with the archive, though, would seem to be less to do with the design task itself, but rather, are a consequence of the wish to use techniques that, as yet, expect examples to be complete, well-described and plentiful.

It would seem likely that much additional information is generated by and around the design task, and, presumably, this too might be a potential source of design knowledge. However, considering the difficulties encountered when collecting examples consisting of the combination of specifications and solutions – which might be thought to be the most 'explicit' of design information - then it would seem unlikely that this additional information could be acquired in quantities sufficient to allow its exploitation by methods similar to those discussed in this thesis.

12.1.4 The Machine-Learning of Design Synthesis Heuristics

Inductive machine learning algorithms offer a way by which the implicit knowledge in the examples might be exploited. Based on a set of examples of some concept, under the right circumstances, these algorithms are able to form a generalised description of the concept, which can then be used to predict a correct response to future events. If design heuristics could be machine-learned from examples of their application, this would provide some of the knowledge required for design KBSs, knowledge that has proved difficult to capture using knowledge engineering techniques. There has been scanty previous research into such an approach, and that which has been reported in the literature is inconclusive, being based, in general, on artificial design problems and data.

Each of the examples of the fluid power circuit design task that have been collected consists of a specification and the corresponding solution. As such, the archive can be considered to contain the knowledge relating aspects of the specification to elements of the solution (*inference* knowledge) but not knowledge of the method by which this inference knowledge is used to arrive at a solution (*strategic* knowledge). This strategic knowledge is also heuristic in nature, and so, difficult to acquire. Nonetheless, in order to form a complete design KBS, this knowledge, along with the *working* knowledge, the final of the categories of persistent knowledge, must be supplied from some other source. In the work reported here, this knowledge has been handcrafted.

This thesis describes five design KBSs that have been constructed using members of three distinct classes of ML algorithm – *artificial neural network*, *classification construction* and *conceptual clustering* algorithms. For each system, then, it has been necessary to propose some strategy by which a specification is transformed into an appropriate solution. From this

strategic knowledge, the necessary inference knowledge can be identified, and as a consequence, the learning tasks can be defined. Since it is heuristic in nature, the accuracy of this handcrafted strategic knowledge is questionable, and an additional and, possibly, malign influence here has been the intention to use particular ML algorithms, which can mean that the strategic knowledge can appear less than natural (as in the case of the specification and solution representations).

Artificial neural networks are able to learn associations between input and output patterns, and as such, would seem at first sight to be the most appropriate algorithms for learning the inference knowledge between specifications and solution elements. However, ANNs operate with numerical data, and since the representations used here are, for the most part, of a qualitative nature, the examples have to be extensively processed so as to encode them in a suitable form. Encoding the data while preserving the information contained within is difficult. A further problem here affects the training procedure and arises due to the relative lack of available examples. The usual method of applying inductive machine learning involves dividing the data into two sets – training data and testing data. The learning process is considered to have been successfully completed when the learned knowledge predicts the values in the testing set with satisfactory accuracy. To have a representative test set drawn from the few available examples would seem unlikely, and so all examples have been devoted to training. This means that it is difficult to decide when training is completed, a particular problem for ANNs. The is due to the random initialisation of ANNs, meaning that outcome of training can vary extensively from one training episode to the next, and to the nature of the learned knowledge, which is impervious to any assessment of its content by inspection. This is, however, a major limitation of all the inductive ML approaches described in this thesis, and suggests that there are simply too few examples available for such an approach to be a practical proposition.

CN2, a classification construction algorithm, belongs to the second class of algorithm that has been investigated. This algorithm learns the knowledge, expressed in the form of rules, which will allow some event to be classified into one of a limited set of groups. This is not a particular natural way of thinking of the process design synthesis; nevertheless, a strategy was devised which involves classifying each of the solution elements as being present in or absent from a solution. This algorithm is able to deal with symbolic data, so no encoding problems were encountered. In contrast to the ANNs, the knowledge learned by CN2 is intelligible, and so can be inspected to discover what has been learned.

An approach adopted by several researchers in the past has been to use a conceptual clustering algorithm to cluster similar design specifications, and then use this as a similarity

measure in a CBR system in order to suggest a solution to a new problem. A similar approach was tried here using the COBWEB algorithm; however, it quickly became apparent that this method is seriously flawed, since the resulting hierarchy merely reflects similarities amongst the examples that happen to be available, and expresses nothing about the design task. For this reason, this approach was dismissed summarily.

The KBSs that incorporate ANN- and CN2-learned inference knowledge are able to produce good designs, seemingly through the application of appropriate heuristics. However, they are also capable of generating poor designs. It seems that the inference heuristics that have been learned by the algorithms are neither wholly correct nor complete enough to respond to the full range of design problems. The incompleteness of this knowledge is apparent in the case of the classification rule KBS, when, for certain problems, none of the rules is applicable to determine whether or not a particular element should be present in a solution. A similar incompleteness is suspected in the heuristics learned by the ANNs, but is difficult to distinguish from incorrect knowledge due to the opacity of the trained networks.

The most obvious reason for the incompleteness and inaccuracy of the machine-learned heuristics would seem to be the lack of training data. However, there does seem to be a more profound problem, which is the lack of *consistency* displayed by the archive examples. Current inductive machine learning algorithms expect training data to be the product of a single, consistent system, which produces well-defined outputs in response to well-defined inputs. This is patently not the case in design contexts, with examples produced by different people at different times for different purposes. Two designers might solve the same problem in two wholly different ways. It seems likely that those relationships that are successfully learnt are those that are well-exemplified and consistently employed throughout the data.

Rather than ascribing the inadequacies of the inference heuristics to the data, though, these problems may well be better attributed to the disparity between the complexity of human learning processes and the unsophisticated state of current machine learning techniques. The investigations into the use of machine learning reported here only partially substantiate the research hypothesis – it does seem possible to acquire some synthesis knowledge from examples in this way, in the form of inference heuristics. However, these heuristics do not appear to be either complete or correct enough to enable the construction of a satisfactory design KBS for the design task considered here. Furthermore, even with a large number of examples, it seems unlikely that current ML algorithms would be able to generate wholly adequate inference knowledge.

12.1.5 Case-Informed Reasoning as Design Synthesis

In order to try to exploit the knowledge implicit in the archive, attention turned to the Case-Based Reasoning approach to design. A major objection to the 'standard' form of this approach arises due to the need to apply modification heuristics in the event of the failure to find an exact match to the current problem amongst the stored examples. With the number of examples available in this instance, this modification stage would seem to be essential, but, once again, these heuristics are not readily available.

If the archive were to be used for CBR, while avoiding as far as possible the problems that knowledge engineering can bring, a different method would have to be developed. Such a method has been developed - *Case-Informed Reasoning*, a constructive CBR approach. Rather than retrieving and then adapting a complete solution, only useful *elements* of solutions are retrieved, which are then combined to form the new design solution. The identification of these useful elements requires further knowledge, in the form of *explanation rules*. These rules express analytical knowledge about the domain – they allow an analysis of the functionality of a given circuit to be made. They are not synthesis heuristics, which would be used in the 'opposite' sense to suggest a circuit on the basis of some given functionality – and this fact would seem to make them more amenable to acquisition through knowledge engineering. In the CIR method, for each required functional attribute in a new specification, the most similar previous design example in which this attribute was successfully satisfied is found. The rules are then used to determine how the function was satisfied in this example – and these elements are then suggested as being appropriate for satisfying that function in the current case. This is done for all of the required functions, after which, the set of selected elements, along with the template, is the solution to the current specification.

The introduction of further domain knowledge (in the form of the explanation rules and the classification of attributes) into the model can be seen as one way of addressing the lack of examples – it provides an environment in which to capitalise upon the knowledge implicit in the archive. The inconsistency of examples ceases to be so great a problem, since there is no obligation to integrate the information contained within them into coherent design bodies of inference heuristics, as was attempted with the use of inductive ML algorithms.

This model can be seen as implementing abductive reasoning for design. Decisions are made based on the combination of the 'definite' knowledge of the explanation rules, the similarity measure and the subjective experience of successful design episodes. The 'most likely' hypotheses about how to satisfy the individual functions are used to generate a complete

circuit hypothetical 'cause' of the specification 'effect'. A degree of completeness of this reasoning is provided by the fact that there will always be a 'most similar' example in the archive (even if this example is, in actuality, quite dissimilar).

The CIR model of design would seem to confirm the research hypothesis, albeit under controlled conditions of a restricted conceptual design task, and a restricted description of this task. The system implementing the model can access the synthesis knowledge implicit in previous designs, and re-apply it to solve new conceptual design problems. However, in order to do so, it has been necessary to introduce external task knowledge into the system. Furthermore, the reliance on the existence and availability of this knowledge would seem to mean that this approach is not applicable to all conceptual design tasks, nor to all configuration design tasks. The system also exploits the simplistic nature of the static state specification representations in the similarity measure adopted. A more complex representation would presumably require a more complex measure of the similarity of specifications.

12.2 Future Work

The research has proven to be successful in its aims. Conceptual design is a complex and difficult task, and the processes involved are not fully understood. An analysis of the logic of design allows some progress to be made towards emulating aspects of the performance of design, and the research has shown that it is possible to use knowledge implicit in designs to solve new problems. However, the CIR method developed here works only for a restricted task, and if the scope were to be widened, much work would be necessary. The larger question of whether a truly automated design system, having all the abilities of a human designer, could ever be constructed for this, or any other similarly complex design task remains unanswered.

12.2.1 Short Term Research

The research discussed in this thesis deals with a constrained design problem, that of configuring single pump-single actuator circuits. The current representation of the problem is necessarily rather limited. In trying to capture the essentials of the conceptual design task, the specification and solution representations are focused upon the expression of functionality. For the specification in particular, this expression is quite limited, and could well be developed. Furthermore, this description could be extended to include reference to performance levels, spatial, noise and other constraints, costs and so on, factors which would seem to have a bearing on the solution at the level at which it is considered here. Each

attribute in the current specification representation can take one of a limited set of qualitative values. This level of expression might be improved to incorporate quantitative terms, and richer descriptions of the qualitative terms. The current solution representation is restricted to describing circuits based on a single template and using members of a limited set of solution elements, a representation which could be extended to include more circuits. Overall, then, there is much potential for enriching these representations.

However, for the CIR system to then use these enhanced representations, it would be necessary to re-describe the existing archive accordingly, and collect more (and more richly described) examples to cater for the greater range of potential problems. As seen earlier, this is not a straightforward task.

The possible extension of the representations and the archive raises some questions about the CIR algorithm itself. If additional circuit templates were to be included in the solution representation, this would require that the algorithm be modified to include reasoning about which template to use in a particular context. The similarity measure currently used is a simple one, and takes advantage of the simple nature of expressing specifications; if the scope of the representations were to be widened to incorporate, for example, quantitative terms, then the measure of similarity would have to be modified accordingly. For the purposes of this research, a domain-independent measure has been applied, but, presumably, better solutions would follow from using a better measure. The source of this better measure presents a problem, however, since it would seem to be quite complex heuristic knowledge in its own right. Most CBR approaches seem to adopt the sort of simple measures seen here. An investigation into the possibilities of acquiring this knowledge, either through knowledge engineering, or, perhaps, machine learning, might be fruitful.

Additional knowledge might also be introduced to deal in a more sophisticated manner with the process of combining the selected solution elements into a coherent solution, and thereby avoiding the problem of over-engineered solutions. The explanation rules could also be enhanced to include more information. It might prove profitable to investigate further the acquisition of these rules through the use of machine learning techniques, an idea introduced in section 11.3.2. If successful, this would offer an appealing approach whereby the archive is exploited to acquire these rules, and later exploited a second time to construct new solutions using the rules.

The consideration of the fluid power systems design task in chapter 5 suggested that the generation of solutions might not proceed through sequential stages of conceptual, embodiment and detail design. Instead, for particular elements of the solution, quite detailed

decisions are made early in the process before returning to complete the conceptual design. This sort of process might be essential for the successful construction of solutions, and the investigation of a similar strategy within the CIR model it may prove of benefit. Moreover, the scope could be widened to investigate whether the CIR model is an appropriate one for the design of fluid power *systems*, rather than single circuits – again, this would require more sophisticated representations and more archive examples.

As described in section 11.3, there is much that could be done to make the CIR system more sophisticated, and a 'better' design tool, if this were thought to be worthwhile. These improvements range from software development, such as adding a more informative user-interface to the system, to developing the algorithm so that it produces better solutions more often. One way in which this development might be achieved is by 'tutoring' the system, that is, asking an expert to use the system to check the answers to sample design problems. In response to a poor solution, the expert would be required to suggest a modification of the system's knowledge (for example, the addition or removal of archive solutions or explanation rules) that would prevent similar poor behaviour occurring in the future. After a certain amount of this form of tutoring, the system might be considered to have obtained a level of reliability which would make realistic its use as a means of suggesting initial solutions in organisations where expertise is otherwise lacking.

For each of the intelligent design systems constructed in the course of this research, it has been necessary to handcraft the required strategic knowledge. Since this knowledge represents heuristic design expertise, handcrafting by a non-design expert would seem to be particularly unsatisfactory, and knowledge engineering approaches, as seen, have their limitations. Thought might be given to alternative potential sources of these design strategies, and the ways in which this knowledge might be automatically acquired from these sources.

12.2.2 Long Term Research

The current level of understanding of design processes is low. There would seem to be much fundamental research into the nature of engineering design, and of designers, still to do. The understanding of the logic underpinning design reasoning is limited and detailed descriptions of the types of knowledge that is needed, and the ways in which it is used is lacking. The development of these ideas would provide a more solid foundation for the construction of automated design systems, which are, as yet, based on a great many assumptions and hypotheses about the design process. The logic of design synthesis is vague – the ideas presented here in chapter 2 suggest a procedure by which configuration designs are

developed, but this would, at the very least, need to be modified to encompass other, original design tasks. The epistemology of design tasks could benefit from more empirical research, but this is difficult and time-consuming, in no small part due to the problems surrounding the articulation of knowledge that this research has attempted to address.

Another area that would benefit from more research is the field of machine learning. The idea of a computer system being able to acquire automatically, from its environment, the skills and knowledge required to perform intelligent tasks remains an appealing one. Current ML algorithms are, on the whole, domain independent and make certain assumptions about their learning tasks, namely that the data are consistent, and accurately and fully described. Little use is made of background domain knowledge, meaning that each learning episode is, effectively, starting from a position of ignorance about the task rather than building upon what is already known. Perhaps domain-specific, or even task-specific, algorithms are necessary.

Indeed, regardless of the methods by which they are constructed, intelligent design systems would stand to gain much from access to a repository of background knowledge, both about the domain in question and of a more general, 'common-sense' nature. There has been little work as yet devoted to the provision of this sort of 'foundational' knowledge in computer systems, and much research effort would seem to be necessary in order to establish its content and representation.

12.3 Summary

The research described in this thesis has been successful in its principal aim of suggesting a manner by which the design synthesis knowledge embodied in examples of previous design episodes can be exploited. To do this, a variant on the AI technique of Case-Based Reasoning, called Case-Informed Reasoning, has been developed. CIR makes use of knowledge about the successful use of particular solution elements in particular contexts that these examples provide.

To conclude, it is worth reiterating a number of the more important general points that have been highlighted in the course of this research:

- A rigorous logic of design synthesis is lacking - the process by which some specification is transformed into the description of a solution is not well understood.

- Successful synthesis is dependent on having and using the appropriate knowledge. This knowledge encompasses, amongst other things, the representation of specifications and

solutions, inference knowledge, and strategies for performing the design task. These types of knowledge differ in form and expression. A characteristic common to all, however, is that they are difficult to acquire and express, with none of the techniques hitherto suggested for doing so being wholly satisfactory.

- In this research, the source of some of this knowledge was to be the archive of examples of previous design episodes. In collecting this material, it was found that the available examples can be few in number and incomplete. That information which was available seemed to suggest itself most readily as a source of inference knowledge. In order to provide the level of consistency and completeness demanded by the available techniques for exploiting these examples, it has been necessary to perform a certain amount of reverse engineering of the examples.

- The inductive ML algorithms that are currently available do not seem to possess the necessary levels of sophistication to enable them to learn inference knowledge from the archive. These algorithms expect problems to be clearly and fully described, and training data to be both representative and the product of a single self-consistent system. It seems unlikely that any of these criteria will be met in the context of design tasks such as that investigated here.

- CBR approaches seem better able to reason using examples from a variety of sources. The CIR development of the basic CBR approach is an attempt to make the most of the few examples of this design task that are available, using additional analytical knowledge of the domain.

The automation of design synthesis is difficult, with no established methodologies for developing computer-based intelligent design systems – and, indeed, there is little indication of the extent to which it is possible at all. While much of this research work has been devoted to gaining a better understanding of the fundamental principles underpinning the design process, there has been a strong emphasis throughout on the implementation of the developed ideas into working, testable computer systems. One of the strengths of AI research is that it encourages the implementation of what might otherwise remain untested theories about intelligent behaviour. These implementations help to confirm those areas in which the theory is sound, and, more often than not, indicate those areas where the theory is lacking. It is hoped that the research presented in this thesis is viewed in the light of a contribution, however small, to both the theory and the practice of design automation.

References

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules, *Proceedings of the Twentieth International Conference on Very Large Databases*, Santiago, Chile, September 1994.

Altshuller, G. (1984). *Creativity as an Exact Science*, Gordon and Breach, New York.

Bachant, J. and McDermott, J. (1984). R1 revisited: four years in the trenches, *AI Magazine*, Fall 1984, pp. 21-32.

Bardasz, T. and Zeid, I. (1993). DEJA VU: case-based reasoning for mechanical design, *Artificial Intelligence for Engineering Design, Analysis and Manufacture*, 7(2), pp. 111-124.

Berry, D. C. (1987). The problem of implicit knowledge, *Expert Systems*, 4(3), pp. 144-151.

Birmingham, W. P., Brennan, A., Gupta, A. P. and Siewiorek, D. P. (1988). MICON: A single-board computer synthesis tool, *IEEE Circuits and Devices Magazine*, January, pp. 37-46.

Boden, M. A. (1987). Artificial Intelligence. In *The Oxford Companion to the Mind*, Gregory, R. L. (ed.), Oxford University Press, Oxford, pp. 48-50.

Boswell, R. (1990). Implementation of the *CN2* algorithm in *C*. Available via *ftp* from: `ftp://ftp.cs.utexas.edu/pub/pclark/cn2.tar.Z`

Bowen, J. A. (1985). Automated configuration of hardware for dedicated microprocessor application systems. In *Knowledge Engineering in Computer-Aided Design*, J. S. Gero (ed.), Elsevier Science Publishers, North-Holland.

Bratko, I. (1993). Machine learning in artificial intelligence, *Artificial Intelligence in Engineering*, 8, pp. 159-164.

Bratko, I. and Muggleton, S. (1995). Applications of inductive logic programming, *Communications of the ACM*, 38(11), pp. 65-70.

Brown, D. (1998). Defining configuring, *Artificial Intelligence for Engineering Design, Analysis and Manufacture*, 12(2), pp.

Brown, D. C. and Chandrasekaran, B. (1986). Knowledge and control for a mechanical design expert system. *IEEE Computer*, 19(7), pp. 92-100.

Brown, D. R. and Hwang, K.-Y. (1993). Solving fixed configuration problems with genetic search, *Research in Engineering Design*, 5, pp. 80-87.

BS 7000 (1989). *Guide to Managing Product Design*, British Standards Institute, London.

Buchanan, B. G. and Shortliffe, E. H. (eds.) (1984). *Rule-Based Expert Systems*, Addison-Wesley, Reading, MA.

Bylander, T., Allemang, D., Tanner, M. and Josephson, J. (1991). The computational complexity of abduction, *Artificial Intelligence*, 49, pp. 25-60.

Carbonell, J. G., Michalski, R. S. and Mitchell, T. M. (1983). An overview of machine learning. In *Machine Learning: An Artificial Intelligence Approach*, Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (eds), Tioga Publishing, CA., USA.

Charniak, E. and McDermott, D. (1985). *Introduction to Artificial Intelligence*, Addison-Wesley, Reading, Mass.

Clancey, W. J. (1986). Heuristic classification. In *Knowledge-Based Problem Solving*, J. S. Kowalik (ed.), Prentice-Hall, New Jersey.

Clark, P. and Boswell, R. (1991). Rule induction with CN2: some recent improvements. In *Machine Learning — Proceedings of the Fifth European Conference (EWSL-91)*, Kodratoff, Y. (ed.). Springer-Verlag, Berlin, pp. 151–163.

Clark, P. and Niblett, T. (1989). The CN2 induction algorithm, *Machine Learning*, 3(4), pp. 261–283.

Court, A. W. and Potter, S. (1996). Fluid power systems and circuits archive, *Internal Report no. 19/96*, Department of Mechanical Engineering, University of Bath, Bath, UK.

Coyne, R. (1988). *Logic Models of Design*, Pitman, London.

Coyne, R. D., Newton, S. and Sudweeks, F. (1993). A connectionist view of creative design reasoning. In *Modeling Creativity and Knowledge-Based Creative Design*, Gero, J. S., and Maher, M. L. (eds.), Lawrence Erlbaum, New Jersey, pp. 177–209.

Coyne, R. D., Rosenman, M. A., Radford, A. D., Balachandran, M. and Gero, J. S. (1990). *Knowledge-Based Design Systems*, Addison-Wesley, Reading, MA.

Cross, N. (1994). *Engineering Design Methods*, John Wiley & Sons, Chichester, England.

Darlington, M. J. (1998). Problem reduction in complex domains. *Internal Report no. 52/98*, Department of Mechanical Engineering, University of Bath, Bath, UK.

Darlington, M. J. and Potter, S. (1998). Fluid power systems design archive, *Internal Report no. 49/98*, Department of Mechanical Engineering, University of Bath, Bath, UK.

da Silva, J. C. and Dawson, D. (1997). The development of an expert system for hydraulic systems design focusing on concurrent engineering aspects, *Proceedings of the 11th International Conference on Engineering Design*, Tampere, Finland, volume 2, pp. 271-276.

Dolsak, B. and Muggleton, S. (1992). The application of inductive logic programming to finite-element mesh design. In *Inductive Logic Programming*, Muggleton, S. (ed.), Academic Press, San Diego, Ca., pp. 453-472.

Duffy, A. H. B. (1997). The 'what' and 'how' of learning in design, *IEEE Expert Magazine*, May/June 1997, pp. 71–76.

Duffy, S. M. and Duffy, A. H. B. (1996). Sharing the learning activity using intelligent CAD, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 10(2), pp. 83–100.

Ericsson, K. A. and Simon, H. A. (1984). *Protocol Analysis: Verbal Reports as Data*, MIT Press, Boston.

Finger, S. and Dixon, J. R. (1989). A review of research in mechanical engineering design. Part I: descriptive, prescriptive, and computer-based models of design processes, *Research in Engineering Design*, 1, pp. 51-67.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering, *Machine Learning*, 2(7), 139–172.

Fox, J. (1996). Expert systems and theories of knowledge. In *Artificial Intelligence*, Boden, M. A. (Ed.), Academic Press, San Diego, Ca.

FPC, (1997). Introduction to hydraulic circuits and components, Fluid Power Course Module FP1, Fluid Power Centre, University of Bath.

Frayman, F. and Mittal, S. (1987). COSSACK: A constraints-based expert system for configuration tasks, *Proceedings of the 2nd International Conference on Applications of AI to Engineering*, Boston, MA., pp. 143-166.

French, M. J. (1985). *Conceptual Design for Engineers*, 2nd Edition, Springer-Verlag, London.

Gennari, J. H., Langley, P. and Fisher, D. (1989). Models of incremental concept formation, *Artificial Intelligence*, 40, pp. 11-61.

Gero, J. S. (1990). Design prototypes: a knowledge representation schema for design, *AI Magazine*, 11(4), pp. 26-36.

Giarratano, J. C., and Riley, G. (1997). *Expert systems: principles and programming*, 3rd Edition, PWS Publishing, Boston, MA.

Gillies, D. (1996). *Artificial Intelligence and Scientific Method*, Oxford University Press, Oxford.

Goel, A., Bhatta, S. and Stroulia, E. (1997). KRITIK: an early case-based design system. In: *Issues and Applications of Case-Based Reasoning in Design*, Maher, M.L. and Pu, P. (eds.), Lawrence Erlbaum, 1997, pp. 87-132.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Mass.

Goodman, N. (1983). *Fact, Fiction and Forecast*, 4th Edition, Harvard University Press, London.

Habermas, J. (1978). *Knowledge and Human Interests*, 2nd Edition, Heinemann, London.

Harman, G. (1965). The inference to best explanation, *Philosophical Review*, 74, pp. 88-95.

Hart, A. (1988). Knowledge acquisition for expert systems. In *Knowledge, Skill and Artificial Intelligence*, Göranzon, B. and Josefson, I. (eds.), Springer-Verlag, London.

Hayes-Roth, F. and Jacobstein, N. (1994). The state of knowledge-based systems, *Communications of the ACM*, 37(3), pp. 27-39.

Henke, R. W. (1983). *Fluid Power Systems and Circuits*, Penton Publishing, Cleveland, OH.

Hua, K. and Faltings, B. (1993). Exploring case-based building design — CADRE, *Artificial Intelligence for Engineering Design, Analysis and Manufacture*, 7(2), pp. 135-143.

Hume, D. (1975). *An Enquiry Concerning Human Understanding*. In, *Hume's Enquiries*, Selby-Bigge, L. A. (ed.) 3rd edition, Clarendon, Oxford.

Ivezic, N. and Garrett, J. H. Jr. (1994). A neural network-based machine learning approach for supporting synthesis, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 8(2)*, 143–161.

Josephson, J. R. and Josephson, S. G., Eds. (1994). *Abductive Inference*, Cambridge University Press, Cambridge.

Kelly, G. A. (1955). *The Psychology of Personal Constructs*, Norton, New York.

Kota, S. and Lee, C.-L. (1993a). General framework for configuration design: part 1 – methodology, *Journal of Engineering Design*, 4(4), 277-289.

Kota, S. and Lee, C.-L. (1993b). General framework for configuration design: part 2 – application to hydraulic systems configuration, *Journal of Engineering Design*, 4(4), 291-303.

Leake, D. B. (1993). Focusing construction and selection of abductive hypotheses, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France, Morgan Kaufmann, pp. 24-29.

Lenat, D. B. (1983). The role of heuristics in learning by discovery: three case studies. In *Machine Learning: An Artificial Intelligence Approach*, Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), Tioga Publishing, CA., USA.

Lenat, D. B. (1995). Cyc: a large-scale investment in knowledge infrastructure, *Communications of the ACM*, 38(11), pp. 33-38.

Lin, Z.-C. and Shen, C.-C. (1995). An investigation of an expert system for hydraulic circuit design with learning, *Artificial Intelligence in Engineering*, 9, pp. 153-165.

Lippmann, R. P. (1987). An introduction to computing with neural nets, *IEEE Acoustics, Speech and Signal Processing Magazine*, 4(2), pp. 4–22.

Lu, S. C.-Y. and Chen, K. (1987). A machine learning approach to the automatic synthesis of mechanistic knowledge for engineering decision making, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 1(2), pp. 109–118.

Maher, M. L., Balachandran M. B. and Zhang, D. M. (1995). *Case-Based Reasoning in Design*, Lawrence Erlbaum Associates, Mahwah, NJ.

Maher, M.L. and Gomez de Silva Garza, A. (1997). Case-based reasoning in design, *IEEE Expert Magazine*, March-April 1997, pp. 34-41.

Maher, M. L. and Li, H. (1994). Learning design concepts using machine learning techniques, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 8(2), pp. 95–111.

Maher, M.L., and Zhang, D.M. (1993). CADSYN: A case-based design process model, *Artificial Intelligence for Engineering Design, Analysis and Manufacture*, 7(2), pp. 97-110.

March, L. (1976). The logic of design and the question of value. In *The Architecture of Form*, March, L. (ed.), Cambridge University Press, Cambridge.

Marcus, S., Stout, J. and McDermott, J. (1988). VT: An expert elevator designer that uses knowledge-based backtracking, *AI Magazine*, 9(1), pp. 95-112.

McDermott, J. (1982). R1: A rule-based configurer of computer systems, *Artificial Intelligence*, 19(1), pp. 39-88.

Medland, A. J. and Mullineux, G. (1993). A constraint approach to feature-based design, *International Journal of Computer Integrated Manufacturing*, 6, pp. 34-38.

Michie, D. (1973). Knowledge engineering, *Cybernetics*, 2, pp. 197-200.

Michie, D. (1982). The state of the art in machine learning. In *Introductory Readings in Expert Systems*, Michie, D. (ed.), Gordon and Breach, New York, pp. 208-229.

Minsky, M. L. (1975). A framework for representing knowledge. In *The Psychology of Computer Vision*, P. H. Winston (ed.), McGraw-Hill, New York.

Mitchell, T. M. (1999). Machine learning and data mining, *Communications of the ACM*, 42(11), pp. 30-36.

Mittal, S., Dym, C. L. and Morjaria, M. (1986). PRIDE: An expert system for the design of paper handling systems, *IEEE Computer*, 19(7), pp. 102-114.

Mittal, S. and Frayman, F. (1989). Towards a generic model of configuration tasks, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 1395-1401.

Mooney, R. (1991). Implementation of the *COBWEB* algorithm in *CommonLisp*. Available via *ftp* from: `ftp://ftp.cs.utexas.edu/pub/mooney/ml-progs/cobweb.lisp`

Muggleton, S. (1991). Inductive logic programming, *New Generation Computing*, 8(4), pp. 295-318.

Muggleton, S. and Feng, C. (1992). Efficient induction of logic programs. In *Inductive Logic Programming*, Muggleton, S. (ed.), Academic Press, San Diego, Ca., pp. 281-298.

Navinchandra, D., Sycara, K.P. and Narasimhan, S. (1991). A transformational approach to case-based synthesis, *Artificial Intelligence for Engineering Design, Analysis and Manufacture*, 5(1), pp. 31-45.

Newell, A. (1982). The knowledge level, *Artificial Intelligence*, 18, pp. 87-127.

Newell, A and Simon, H. A. (1976). Computer science as empirical inquiry: symbols and search. Reprinted in: *Computation and Intelligence: Collected Readings*, Luger, G. F. (ed.), MIT Press, Cambridge, MA., 1995, pp. 91-119.

Pahl, G. and Beitz, W. (1996). *Engineering Design – A Systematic Approach*, 2nd Edition, Springer-Verlag, London.

Partridge, D. (1996). Representation of knowledge. In *Artificial Intelligence*, Boden, M. A. (ed.), Academic Press, San Diego, Ca.

Peirce, C. S. (1940). Abduction and induction. In *The Philosophy of Peirce: Selected Writings*, Bulcher, J. (ed.), Harcourt, Brace and Co, New York.

Pu, P. (1993). Issues in case-based design systems, Artificial Intelligence for Engineering Design, Analysis and Manufacture, 7(2), pp. 79-85.

Pugh, S. (1991). *Total Design*, Addison-Wesley, Wokingham, England.

Punch, W. F., Tanner, M. C., Josephson, J. R. and Smith, J. W. (1990). Peirce: a tool for experimenting with abduction, *IEEE Expert Magazine*, October 1990, pp. 34-44.

Purvis, L. and Pu, P. (1998). COMPOSER: a case-based reasoning system for engineering design, Robotica, 16, pp. 285-295.

Quillian, M. R. (1968). Semantic memory. In *Semantic Information Processing*, M. L. Minsky (ed.), MIT Press, Cambridge, MA., USA.

Quinlan, J. R. (1986). Induction of decision trees, *Machine Learning*, 1, pp. 81-106.

Reich, Y. and Fenves, S. J. (1989). The potential of machine learning techniques for expert systems, *Artificial Intelligence in Engineering Design, Analysis and Manufacture*, 3(3), pp. 175-193.

Reich, Y. and Fenves, S. J. (1992). Inductive learning of synthesis knowledge, *International Journal of Expert Systems: Research and Applications*, 5(4), pp. 275–297.

Reich, Y. and Fenves, S. J. (1995). A system that learns to design cable-stayed bridges, *Journal of Structural Engineering*, 121(7), pp. 1090–1100.

Reich, Y., Konda, S. L., Levy, S. N., Monarch, I. A. and Subrahmanian, E. (1993). New roles for machine learning in design, *Artificial Intelligence in Engineering*, 8, pp. 165–181.

Rich, E. and Knight, K. (1991). *Artificial Intelligence*, 2nd Edition, McGraw-Hill, N.Y.

Riesbeck, C. K. and Schank, R. C. (1989). *Inside Case-Based Reasoning*, Lawrence Erlbaum Associates, Hillsdale, NJ.

Roderman, S. and Tsatsoulis, C. (1993). PANDA: a case-based system to aid novice designers, *Artificial Intelligence for Engineering Design, Analysis and Manufacture*, 7(2), pp. 125-133.

Roozenburg, N. F. M. (1993). On the pattern of reasoning in innovative design, *Design Studies*, 14(1), pp. 4-18.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Rumelhart, D. E. and McClelland, J. L. (eds.), Vol. 1, pp. 318–362. MIT Press, Cambridge, MA.

Russell, B. (1912). *The Problems of Philosophy*, Oxford University Press, Oxford.

Russell, S. (1996). Machine learning. In *Artificial Intelligence*, Boden, M. A. (ed.), Academic Press, San Diego, CA., pp. 89-133.

Schmidt, L. C. and Cagan, J. (1995). Recursive annealing: a computational model for machine design, *Research in Engineering Design*, 7, pp. 102-125.

Schreiber, G., Wielinga, B. and de Hoog, R. (1994). CommonKADS: a comprehensive methodology for KBS development, *IEEE Expert Magazine*, December 1994, pp. 28–37.

Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: MYCIN*, American Elsevier, New York.

Simon, H. A. (1983). Why should machines learn? In *Machine Learning: An Artificial Intelligence Approach*, Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (eds.), Tioga Publishing, CA., USA.

Smith, J., Svirbely, J. R., Evans, C. A., Strohm, P., Josephson, J. R. and Tanner, M. C. (1985). RED: A red-cell antibody identification expert module, *Journal of Medical Systems*, 9(3), pp. 121-138.

Stillings, N. A., Feinstein, M. H., Garfield, J. L., Rissland, E. L., Rosenbaum, D. A., Weisler, S. E. and Baker-Ward, L. (1987). *Cognitive Science: An Introduction*, MIT Press, Cambridge, MA., USA.

Sullivan, J. A. (1982). *Fluid Power*, 2nd Edition, Reston Publishing Company, Reston, VA.

Takeda, H. (1994). Abduction for design. In *Formal Design Methods for CAD*, J. Gero and E. Tyugu (eds.), IFIP Transactions B-18, Elsevier Science Publishers BV, Amsterdam, pp. 221-244.

Thompson, C. A. and Mooney, R. J. (1994). Inductive learning for abductive diagnosis, *Proceedings of the Twelfth National Conference on Artificial Intelligence*, MIT Press, pp. 664-669.

Ullman, D. G. (1997). *The Mechanical Design Process*, 2nd Edition, McGraw-Hill, New York.

Ulrich, K. T. and Seering, W. P. (1989). Synthesis of schematic descriptions in mechanical design, *Research in Engineering Design*, 1, pp. 3-18.

VDI 2221 (1987). *The Systematic Approach to the Design of Technical Products*. VDI-Verlag, Düsseldorf.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks, *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(3), pp. 328–339.

Welch, R. V. and Dixon, J. R. (1994). Guiding conceptual design through behavioral reasoning, *Research in Engineering Design*, 6, pp. 169-188.

Westman, R., Sargent, C. and Burton, R. (1987). A knowledge-based modular approach to hydraulic circuit design, *Computers in Engineering*, 1, pp. 37-41.

Wielinga, B. and Schreiber, G. (1997). Configuration design problem solving, *IEEE Expert Magazine*, March-April 1997, pp. 49-56.

Winston, P. H. (1993) *Artificial Intelligence*, 3rd edition, Addison-Wesley, Mass., USA.

Zell, A. *et al.* (1996). *SNNS: Stuttgart Neural Network Simulator, User Manual*, version 4.1, Institute for Parallel and Distributed High Performance Systems, University of Stuttgart, Germany.

Zeng Y. and Cheng G. D. (1991). On the logic of design, *Design Studies*, 12(3), pp. 137-141.

A The Archive of Design Examples

This appendix summarises in tabular form the content of the second version of the archive of 30 design examples that was constructed and used in the course of this research (Darlington and Potter, 1998). This is followed by three examples drawn from the archive, presented here with the intention of illustrating something of the nature of the archive document itself.

A.1 The Archive – Summary of Content

Table A-1 displays the following:

- the *source* of each example - the examples were from either internal (Department of Mechanical Engineering course notes) or external sources. In the case of the latter, some further information about the nature of the source has been provided.

- the *completeness of the source specification* – to give some idea of the amount of reverse-engineering and interpretation (see chapter 5) that was required to express each example in an adequate fashion for the purposes of the research, the following codes are used:

    a) complete description, no reverse-engineering required;

    b) adequate description, minimal reverse-engineering;

    c) minimal description, high degree of reverse-engineering;

    d) little or no description, very high degree of reverse-engineering.

- the *confidence in the correctness of the solution* – an indication of the degree of faith that the given circuit represents a good solution to the corresponding (reverse-engineered) specification.

Table A-2 and Table A-3 contain the specification and solution information respectively that was used in both the machine-learning experiments and the implementation of the case-informed reasoning methodology, as described in the main body of this thesis. In presenting this information, the intention is to provide some indication of the range of the available

examples, and (as is particularly evident in the table listing the solution information) the limited coverage of the possible designs that this archive provides.

Table A-2 lists the design specifications of the archive examples, described according to the static state representation introduced in chapter 5. In describing the attribute values in this table, the following key applies:

l – low
m – medium
h – high
nh – non-horizontal
ho – horizontal
y – yes
n – no

Table A-3 describes the solutions to the archive examples, expressed according to the solution representation presented in chapter 5. A tick indicates that the corresponding solution element is present in the solution, a cross that it is absent.

archive example	*source*	*Completeness of source specification (see key)*	*Confidence in correctness of solution*
1	internal	b	high
2	external (company)	c	high
3	external (company)	c	high
4	external (training manual)	c	high
5	external (training manual)	b	high
6	external (text book)	c	high
7	external (text book)	c	high
8	external (text book)	c	low
9	internal	b	high
10	internal	a	low
11	internal	b	low
12	external (training manual)	b	high
13	external (training manual)	a	high
14	external (training manual)	d	high
15	external (conference)	a	high
16	external (training manual)	d	high
17	internal	a	low
18	internal	b	low
19	internal	a	low
20	external (training manual)	c	high
21	external (training manual)	b	high
22	external (company)	a	high
23	external (company)	a	high
24	external (company)	c	high
25	external (company)	a	high
26	external (training manual)	c	high
27	external (training manual)	c	low
28	external (training manual)	c	high
29	external (company)	a	high
30	external (company)	a	high

Table A-1. The sources and original quality of the archive examples.

Archive Example	*maximum load/force*	*maximum speed*	*plane of motion*	*continuously variable speed*	*hold load stationary*	*smooth accelerations*	*hold load on failure*	*load-independent speed*	*control extend speed*	*control retract speed*	*solution requires motor*	*control inertia*	*energy efficiency paramount*	*control accuracy*
1	l	m	nh	n	y	n	y	n	n	n	n	y	n	l
2	m	m	nh	y	y	y	y	n	n	n	n	y	n	h
3	h	l	nh	y	y	y	y	n	n	n	n	y	y	l
4	m	m	nh	y	y	n	y	y	n	n	n	y	n	h
5	m	l	nh	y	y	y	n	y	n	n	n	y	n	h
6	l	m	nh	n	y	n	n	n	y	y	y	n	n	l
7	l	m	ho	n	y	n	n	n	y	y	y	n	n	l
8	l	m	ho	y	n	n	n	n	n	n	y	y	y	h
9	l	m	nh	n	y	n	y	n	n	y	n	y	n	l
10	m	m	ho	n	y	n	n	y	y	y	n	n	n	l
11	h	m	ho	y	y	y	y	n	n	n	y	y	n	h
12	m	m	nh	n	y	n	n	n	n	n	n	y	n	l
13	l	l	ho	y	y	y	n	y	n	n	y	y	n	h
14	l	m	ho	y	y	n	n	n	n	n	n	y	n	h
15	m	m	nh	n	y	n	n	n	n	n	n	n	n	l
16	l	h	ho	y	y	n	n	n	n	n	y	y	n	h
17	l	m	nh	y	y	n	n	n	n	n	n	y	n	h
18	m	h	ho	n	y	n	n	y	y	y	n	n	n	l
19	h	m	nh	y	y	n	y	n	n	n	y	y	y	h
20	m	m	nh	y	y	y	n	n	n	n	n	y	n	h
21	m	m	nh	y	y	y	n	y	n	n	n	y	n	h
22	m	m	nh	n	y	n	y	n	n	y	n	n	n	l
23	l	l	ho	n	y	n	n	n	y	y	n	n	n	l
24	m	m	nh	y	y	y	y	n	n	n	y	y	n	h
25	m	h	nh	y	y	y	n	y	n	n	y	y	n	h
26	m	m	nh	n	n	n	n	n	n	n	n	y	n	l
27	l	h	ho	y	n	n	n	y	n	n	y	y	n	h
28	l	h	ho	y	y	n	n	y	n	n	y	y	n	h
29	l	m	nh	y	y	n	n	y	n	n	n	y	n	h
30	l	l	ho	y	y	n	n	y	n	n	n	n	n	h

Table A-2. The specifications of the archive examples.

Archive Example	PRV1_A	PRV1_A&B	POCV_A	CBV1_A	CBV1_A&B	CBV2_A	PCMP_C&PROP_DCV	DECV_A&B	MO_A	MO_B	CVC_A&B	VPCRV_D	PRV2_A&B	VDP_PMP	MOT_ACT	PROP_DCV	CC_DCV
1	✔	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
2	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖
3	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖
4	✖	✖	✔	✖	✖	✔	✔	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖
5	✖	✖	✖	✖	✖	✖	✔	✔	✖	✖	✖	✖	✖	✖	✖	✔	✖
6	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔	✖	✖	✖	✖	✔	✖	✔
7	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔	✖	✖	✖	✖	✔	✖	✔
8	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✔	✔	✖	✖
9	✔	✖	✔	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖
10	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✔
11	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✔	✖	✔	✖	✔	✔	✖
12	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔
13	✖	✖	✖	✖	✖	✖	✔	✔	✖	✖	✔	✖	✔	✖	✔	✔	✖
14	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✔	✔
15	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔
16	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✔	✖	✔	✔	✔
17	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔
18	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✔
19	✖	✖	✔	✔	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔	✔	✖	✖
20	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✔	✖
21	✖	✖	✖	✖	✖	✖	✔	✔	✖	✖	✖	✖	✖	✖	✖	✔	✖
22	✖	✖	✔	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖
23	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔	✖	✖	✖	✖	✖	✖	✔
24	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✔	✖	✔	✔	✖
25	✖	✖	✖	✖	✔	✖	✔	✖	✖	✖	✖	✖	✖	✖	✔	✔	✖
26	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
27	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✔	✖	✖	✖	✔	✔	✖
28	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✔	✖	✔	✖	✔	✔	✔
29	✖	✖	✖	✖	✖	✔	✔	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔
30	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔

Table A-3. The solutions of the archive examples.

A.2 The Archive – Illustrative Examples

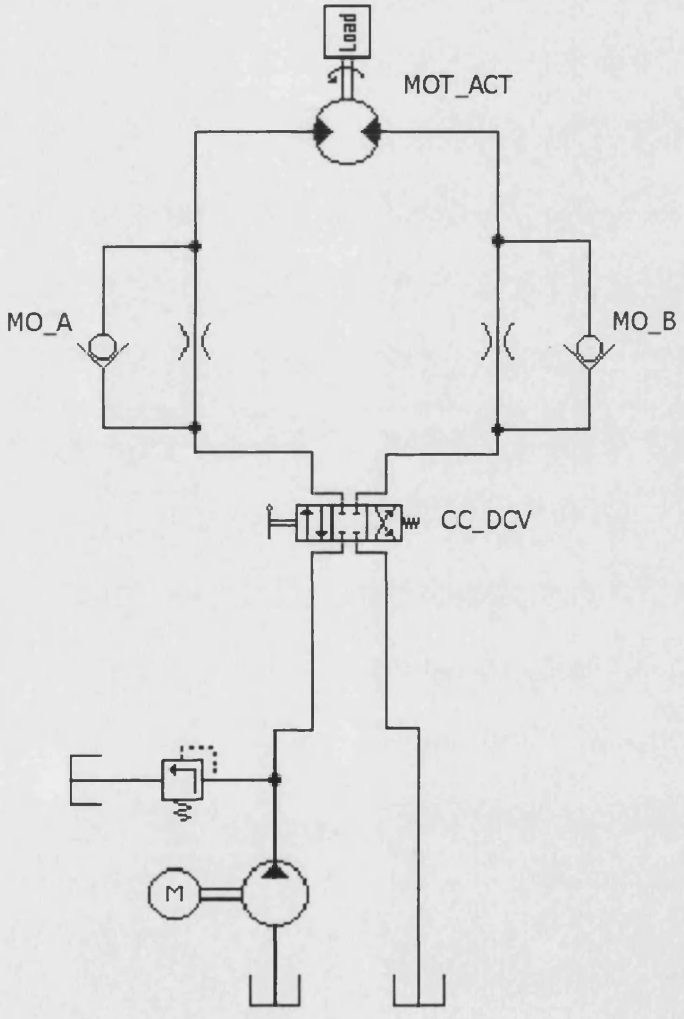
This section contains three of the examples contained in the archive, namely examples 6, 9 and 24. Each example is derived from a different source, and, while each has a well-defined circuit solution, the associated specifications differ in both style and content. These specifications and the solutions were re-described to conform with the devised representations, as shown in Table A-2 and Table A-3 respectively.

ARCHIVE NUMBER: 6	SOURCE: External [text book: (Sullivan, 1982; p. 299)

SPECIFICATION:

"...system to raise and lower a load of 3 tonnes over a distance of 20 feet..."

CIRCUIT SOLUTION:

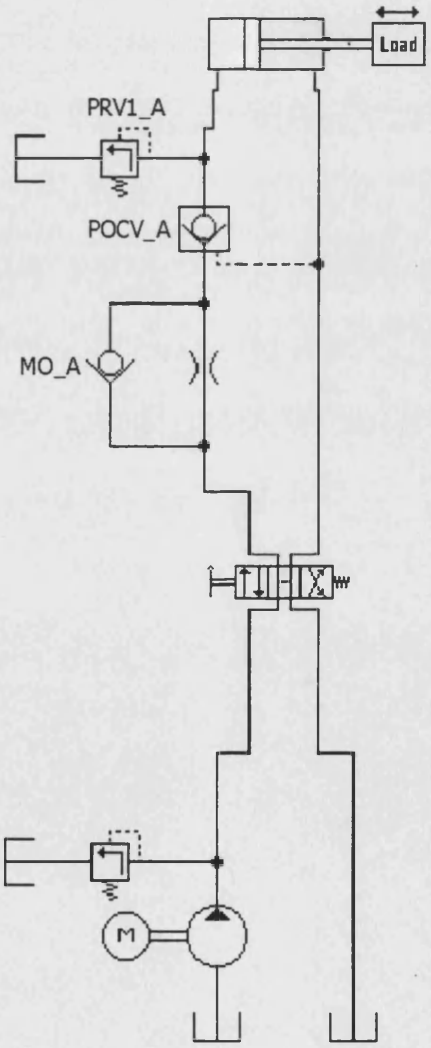


ARCHIVE NUMBER: 9	SOURCE: Internal [University of Bath training course notes]

SPECIFICATION:

"Move a mass of 890 kg vertically up and down over a distance of 0.61m. It is required to stop the mass at any position within the stroke without excessive pressure peaks and avoiding cavitation due to overrun. When held stationary the mass must not be allowed to creep downwards. Control the speed of the mass to enable a quick raise and a slow drop."

CIRCUIT SOLUTION:



ARCHIVE NUMBER: 24	SOURCE: External [fluid power design company]

SPECIFICATION:

Hoist acting vertically. Maximum mass: $1x10^4$kg. Maximum speed: 0.3 ms^{-1}

CIRCUIT SOLUTION: